Design and Implementation of a CAN-USB Interface for Networked Embedded

Constantin Abaceoae
Faculty of Automatic Control and Computer Engineering
"Gheorghe Asachi" Technical University of Iasi
Iasi, Romania
abaceoae.constantin@ac.tuiasi.ro

Mihai Postolache
Faculty of Automatic Control and Computer Engineering
"Gheorghe Asachi" Technical University of Iasi
Iasi, Romania
mpostol@ac.tuiasi.ro

***Abstract*— Controller Area Network (CAN) is the most used protocol for intra-vehicle communication between Electronic Control Units (ECUs) for decades. Moreover, there are several implementations of industrial networks built on top of CAN, (i.e. CANopen and DeviceNet) that expanded even more its area of application to almost any industrial automation field. The growing needs for data in today's modern and safe applications led to CAN Flexible Data Rate (CAN FD) in order to increase the network throughput and pushed the data field of the CAN frames beyond the 1Mbps limit of the high speed CAN chips. A low cost CAN-USB interface application for dual CAN bus network monitoring, diagnose and maintenance is proposed which can be easily ported on any member of the ARM Cortex-M microcontroller family with built-in CAN or CAN-FD communication.**

Keywords*— Controller Area Network, CAN-USB adapter, embedded software application.

I. INTRODUCTION

Since its invention by Robert Bosch in 1983 and its official launching in 1986 [1], the Controller Area Network (CAN) communication protocol has passed through several upgrading (CAN2.0 in 1991) and standardization (as ISO1898 in 1993) steps. A new step ahead was done in 2012 with the new variant CAN Flexible Data Rate (CAN FD), which can extend the data field of a frame from 8 to 64 bytes and has an option to switch to an increased data bitrate, while still remaining compatible with CAN 2.0 frames [2].

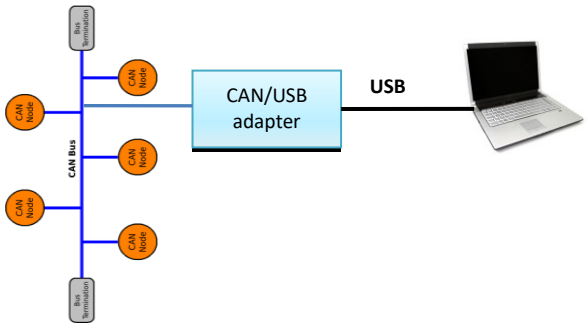

Fig. 1. Use of an off-the-shelf USB-CAN adapter for CAN bus monitoring

After the first standalone CAN chips produced by Intel and Philips in 1987, basic or full CAN controllers were included as internal peripherals by almost any family of microcontrollers for both automotive and industrial markets. Commercial CAN-USB adapters (Fig.1) for connecting mobile devices to CAN are available as black-box solutions that lack flexibility and cannot be customized for application specific requirements.

CAN-USB adapters can easy be implemented using general purpose microcontrollers connected to standalone CAN and USB controllers in a cost-effective embedded system. 8-bit microcontrollers like AT89S51 [3] or AT89C52 [4] get connected to external USB controllers like PDIUSBD12 [3] and FT245BM [4], respectively, and make use of the standalone CAN controller SJA1000 in order to obtain a custom low-cost USB-CAN adapter (Fig. 2).

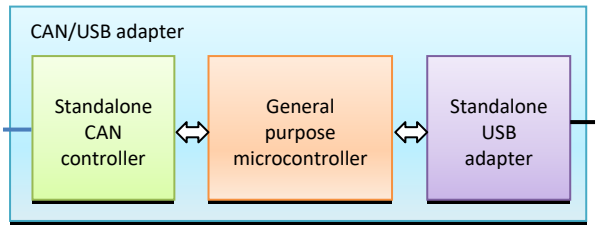

Fig. 2. USB-CAN adapter using standalone USB and CAN bus controllers

In [5] a design of a CAN-USB convertor is proposed, allowing the USB port to be used for the control of CAN devices. A Graphical User Interface (GUI) was developed for testing purposes, based on a custom designed Data Link Library (DLL) Application Programming Interface (API) of virtual serial COM ports.

Our approach (Fig. 3) is to use a 32-bit ARM Cortex-M series microcontroller and its on-chip CAN and USB peripherals to implement a powerful USB-CAN adapter and custom firmware that can be easily ported on a large variety of boards provided with these two communication interfaces. In contrast to [5], the host PC runs a GUI developed using standard DLL libraries provided by the host Windows operating system to demonstrate the CAN-USB gateway easy to use and its simple extension to fit application specific requirements by customization.

A CAN-USB adapter is a useful tool acting as a gateway for any device that has an USB port. It can be used for several applications like CAN bus monitoring or CAN nodes remote configuration, measurement and calibration as presented in [6].

978-1-5386-4444-7/18/$31.00 ©2018 IEEE

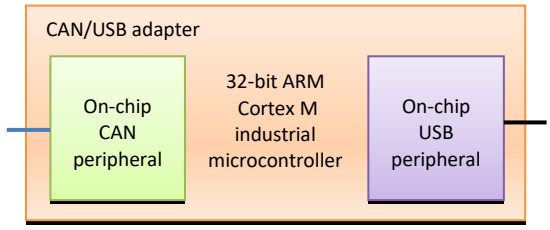

Fig. 3. USB-CAN adapter using integrated USB and CAN bus peripherals

In section II the custom firmware running on an STM32F4DISCOVERY development board is presented together with used resources and schematic diagrams, in section III the implementation and features of the graphical application running on the host computer are described and in the final part experimental tests and results are discussed and conclusions are drawn.

II. The Custom Firmware Application

The development board STM32F4DISCOVERY features an STM32F407 ARM Cortex-M4 168MHz microcontroller core which is connected to dual CAN channels via two external high speed MCP2552 transceivers [7]. The custom firmware was developed using Eclipse GNU ARM GCC toolchain and the STM32CubeMX graphical software configurator tool was used to generate initialization C code for the on-chip peripherals using graphical wizards.

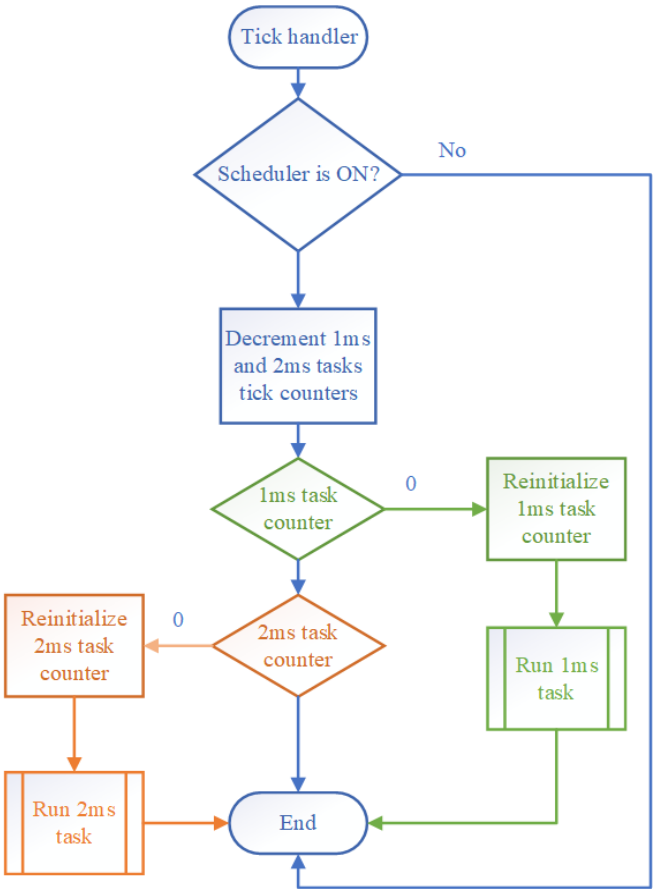

Fig. 4. 10µs time-driven scheduler tick handler

A timer tick is raised every 10µs and act as a time-driven task scheduler. Two tasks are then scheduled each one being activated every 1ms and 2ms, respectively (Fig.4).

A. USB messages dispatching

The 1ms task handles messages received via USB and is responsible for local configuration settings changes as well as with the one shot or cyclic message transmission over CAN, as requested via USB by the graphical user interface.

The command USB message received by the board has the format shown in Fig. 5. There is a command specifier, followed by specific data fields of corresponding lengths.

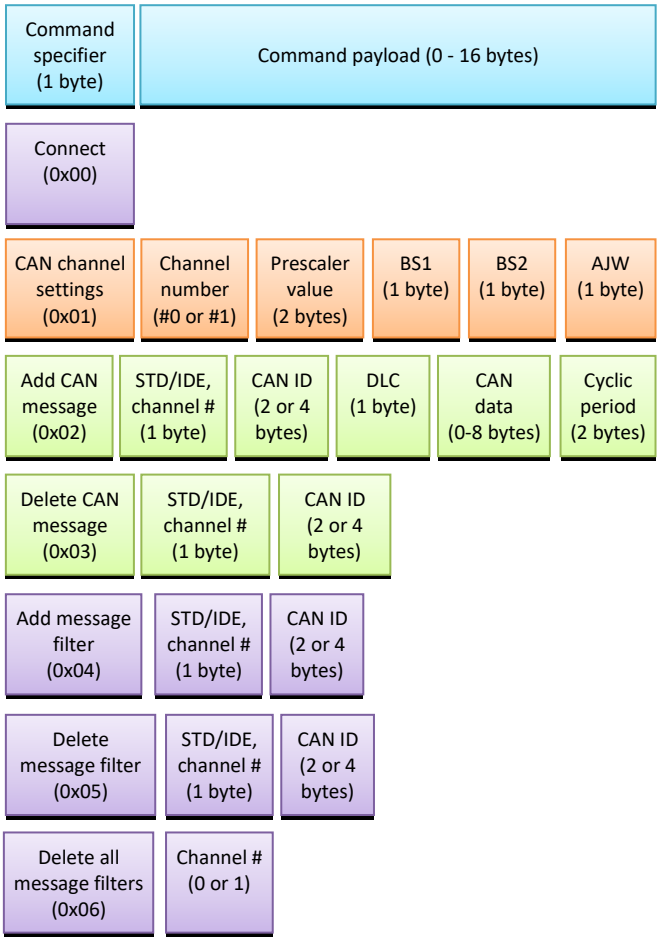

Fig. 5. Command message format of USB received data

The first command type is the CAN configuration settings allowing a number of two CAN channels to be initialized with proper parameter values in order to achieve a specific bit length (and CAN speed): a *prescaler* divider *value* to set the time quantum (*TQ*) derived from the internal oscillator (at 42MHz) as shown in (1) and values for bit segments *BS1* and *BS2* in number of time quanta to establish CAN *BaudRate* (2) as well as the value of the Adjust Jump Width (AJW) number of time quanta used by the CAN controller for hardware resynchronization.

$$TQ = (1/42MHz)/(Prescaler\ Value) \quad (1)$$

$$BaudRate = TQ + TQ_{BS1} \times TQ + TQ_{BS2} \times TQ \quad (2)$$

The second type of command follows the CAN message structure using a standard or extended CAN identifier (CAN ID), the Data Length Code (DLC) field and the CAN payload data of maximum 8 bytes. If the last field is 0 the command will be considered as a one-shot message, otherwise that value gives the number of milliseconds to wait before message has to be periodically retransmitted and the message is put on a waiting list.

The 1ms task decrements the counters of the cyclic messages on the waiting list and activates a flag for those ready to be sent. In the main loop of the application all the flags of the cyclic messages are continuously checked and associated CAN message transmission is triggered for transmission when it is the case.

If the CAN ID of the message is already in the waiting list, its content will be updated. A status flag is used to lock a message entry in the waiting list while its content is updated. One or all messages can be deleted from the waiting list if the value 0x03 is used as command specifier.

Finally, the last three command messages applies or removes a mask for received CAN messages in order to filter the specified CAN ID based on a list of incoming CAN messages that should not be stored and reported.

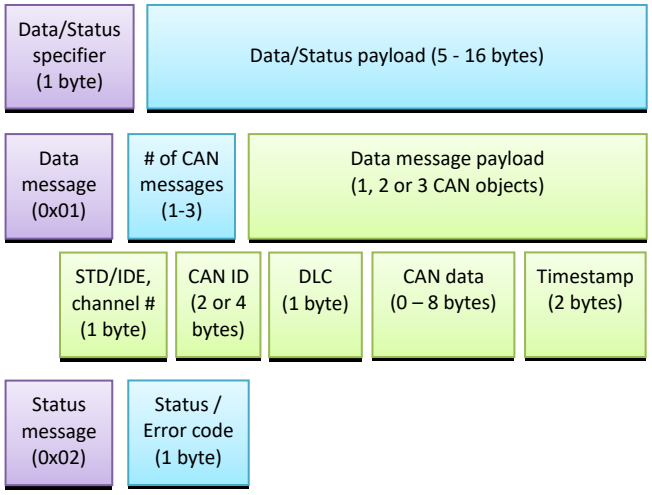

Fig. 6. Data/Status/ message format of USB transmit buffer

Received CAN messages are put in a cyclic queue by the CAN interrupt handler, from which the 2ms task takes at most three of them at once to be copied in the USB buffer using Direct Memory Access (DMA). At the end of DMA transfer the buffer content (Fig. 6) is automatically sent to the graphical interface via USB.

B. *CAN message buffering and processing*

CAN messages are received in the CAN interrupt handler that applies the active filters and put the allowed messages into a circular queue. The 2ms task copy a number of maximum three messages using a fast DMA transfer from the queue into the USB buffer (at most 62 bytes), then an USB interrupt is raised at the end of DMA transfer and the status of the circular queue is checked and updated.

Status/Error message codes are sent as response to the USB received command messages or when internal errors occurs while receiving and processing CAN messages.

III. The CAN-Spy Graphical Interface

A graphical interface running on a host Windows PC was designed and implemented in Eclipse using with MinGW GCC and regular Win32 API library functions from the *user32.dll* and *comctl32.dll* dynamic link libraries. A *WinMain* loop is used for receiving, processing and transmitting messages to the main and child windows and their control objects like buttons, edit boxes, and check boxes. The main window of the graphical interface is presented in Fig. 7.

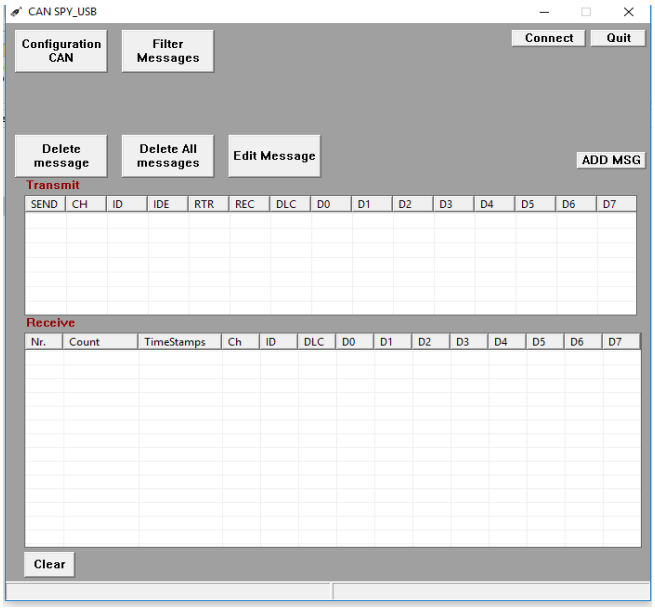

Fig. 7. The main window of the graphical interface

The host PC application was interfaced with the development board for USB communication using the *Winusb.dll* dynamic link library API functions [8].

Configuration, connect and CAN message create, edit, and delete buttons are provided to establish a connection to the development board and to set its CAN controller communication parameters and handle CAN messages in the Transmit window.

All actions in the application can only be done if the application is connected to the development board. This is possible by pressing the *Connect* button, available in the interface.

When a button is pressed, a Windows command message (*WM_COMMAND*) is initiated on the branch for the button with the corresponding *id* in which the connection function is called. The connection is possible by calling the

RawHID_Open() function (Fig. 8). This function receives the *VID* (*vendor ID*) and *PID* (*product ID*) parameters, which must have the same values as the corresponding parameters set by firmware on the USB controller of the development board.

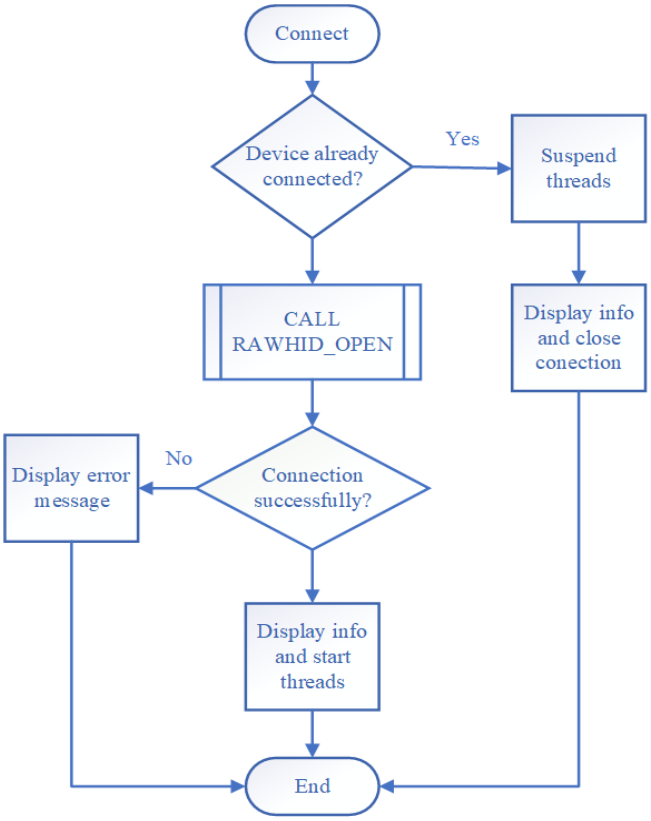

Fig. 8. Connecting the target board as Human Interface Device

Status and error messages are displayed in pop-up windows and bring information about the way a command was received and processed on the connected target board. Also, error messages can notify the user about unwanted events like changing the CAN bus error management status (*Active Error*, *Pasive Error* or *BusOff* status, heavy CAN bus conditions, wrong *BaudRate*), or indicate overrun conditions on both the target USB-CAN adapter or the host PC graphical interface.

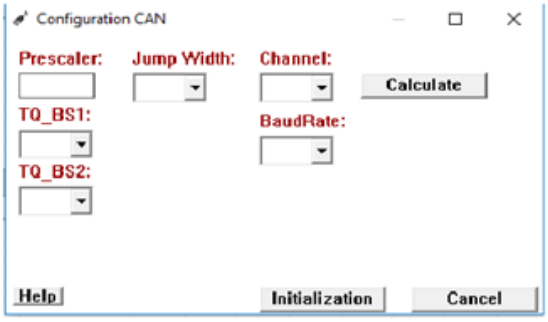

Fig. 9. CAN channel configuration window

A number of two CAN channels can be individually configured as in the child window shown in Fig. 9. The CAN bus *BaudRate* for the selected channel can be calculated from the prescaler, *TQ_BS1* and *TQ_BS2* values, according to relation (2).

Then the Initialization button has to be pressed to have the new communication parameters encapsulated and sent over USB as presented before in Fig. 5.

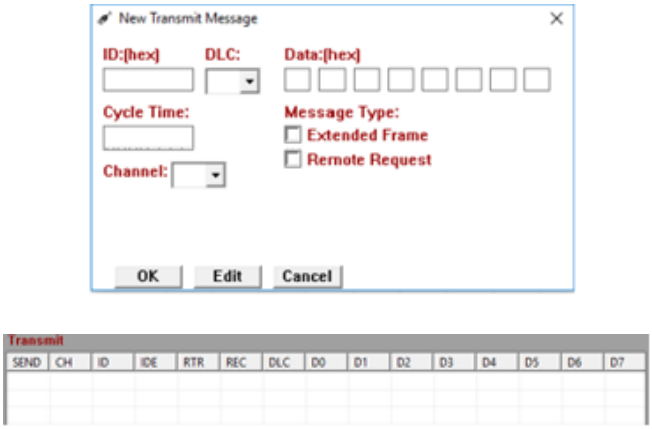

Fig. 10. New message window and transmit section

When a new CAN message is added a child window shown in Fig. 10 is created to retrieve the channel number, CAN ID, DLC and data fields, as well as the recurrence interval expressed in milliseconds. Input data is retrieved to be packaged in a buffer that is automatically set in the Transmit graphical section as ready for transmission.

On the left side of each message in Transmit section there is a checkbox (*SEND* column). Each time it is checked, the message in the transmission buffer is automatically sent to USB.

Also, when the Edit Message button is pressed, the selected message in the transmit section can be updated. The user is allowed to delete a specific message or even all messages that are entered in the Transmit message viewing section.

On the other hand, messages received on USB are retrieved and processed using several threads (Fig. 11). A reception thread deals with downloading messages from the USB buffer and storing them in a circular queue. The large queue size ensures the safety access for retrieving messages by the other threads.

Two other threads pop the messages from queue in parallel and display them in the Receive graphical section (Fig. 12).

Two threads were used when displaying CAN messages because if we run the application on a computer with lower computing power and we want to receive messages at a rate of 1ms, a single thread would not cope and the display would be far behind.

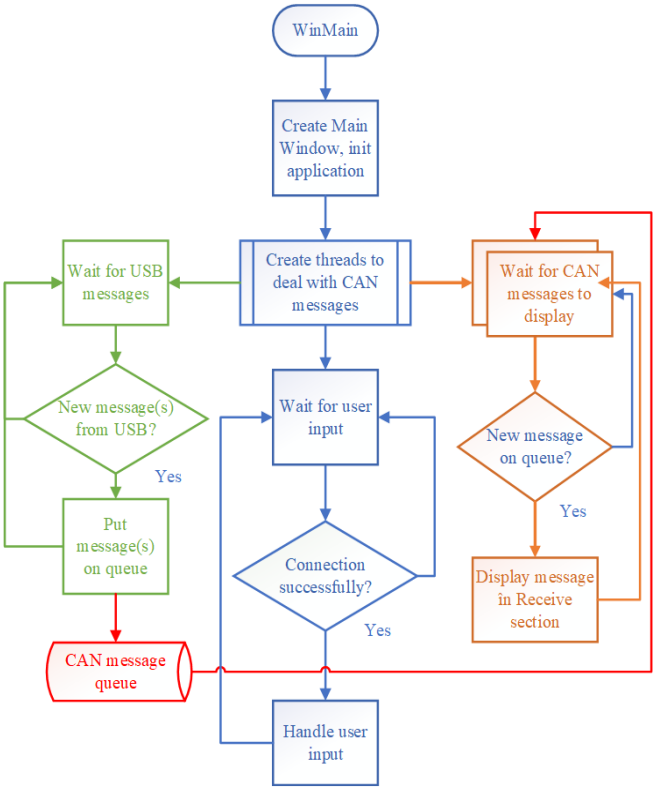

Fig. 11. WinMain multithreading processing

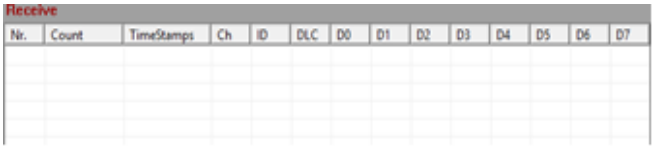

Fig. 12. CAN message Receive graphical section

Pressing the *Filter Messages* button opens a new window (Fig. 13), where one can select which type of ID to be rejected (standard or expanded), the rejected ID, and if a previous filter is canceled on that ID.

Fig. 13. Filter settings window

Single ID or a range of IDs is accepted in both *Add ID* and *Remove ID* edit boxes of the same ID type: standard or extended.

IV. Experimental Results and Conclusions

The proposed CAN-USB adapter operation was tested for both Transmit and Receive way of operation. For this purpose an off-the-shelf CANcaseXL USB-CAN module from Vector was used while PCAN-View from Peak-System was running on another PC.

As it is shown in Fig.14, a message sent from our developed CAN-SPY graphical interface was received in PCAN-View with the correct ID, data and cycle time.

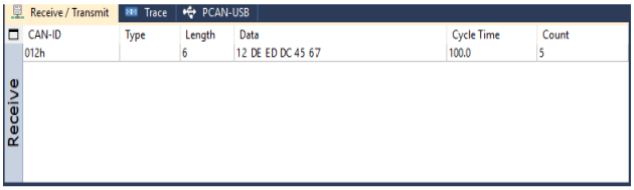

Fig. 14. Receiving a CAN message with PCAN-View

To demonstrate reception operation at 1Mbps with our graphical interface a number of 3 messages were entered in PCAN-view, as shown in Fig.15.

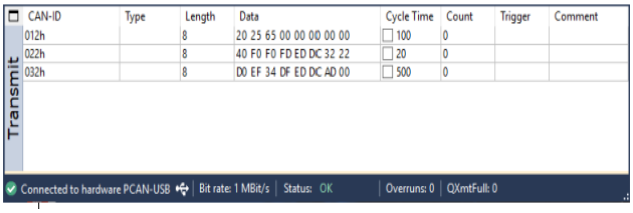

Fig. 15. Testing the Receive operation – messages transmitted with PCAN-View

The corresponding received messages displayed in the Receive section of the CAN-SPY graphical interface are illustrated in Fig. 16.

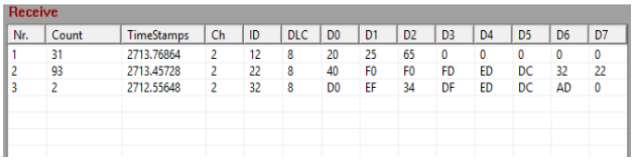

Fig. 16. Testing the Receive operation – messages transmitted with PCAN-View

Compared to the commercial versions of USB-CAN adapters an advantage is that it does not require installation. When copying the executable to any PC, it can run without having to first install drivers or certain frameworks. The development board connected to any PC was seen as a Human Interface Device, being automatically recognized by the operating system or any application that interacts with this module.

Also the software implemented for the STM32F4Discovery board can be directly ported to any other STM32 board, and with few changes to any other ARM Cortex-M series of microcontroller core.

A minimum rate of 1ms is allowed for a burst of messages incoming on the CAN bus in order to be displayed in real time on the Receive graphical section. Depending on the CAN communication bitrate and the network load, messages on CAN bus can occur much faster, but the graphical interface would not handle a larger number of messages timely. An advantage of our USB-CAN adapter software is that it allows user customizations such that selected CAN messages can be monitored for a specific data field only, or a log file in a user defined format may be created locally on a SD card or onto a disk by the host PC interface.

REFERENCES

[1] ISO, ISO 11898-1:2015 – Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling, International Organization for Standardization, 2015

[2] G. Cena, I. C. Bertolotti, T. Hu, A. Valenzano, "CAN XR: CAN with extensible in-frame Reply", IEEE 14th International Conference on Industrial Informatics (INDIN), pp. 1198 – 120, 2016

[3] Y. Luo, "The Design of CAN/USB Embedded Adapter", 3rd International Conference on Mechanical and Electronics Engineering, Hefei, 2011, in Mechanical and Electronics Engineering III, PTS 1-5, Book Series: Applied Mechanics and Materials, Volume: 130-134, pp. 3938 – 3941, 2012

[4] M. Liping, Z. Weiguo, "Design and implementation of USB and CAN communication adapter", Proceedings of International Conference on Intelligent Computation and Industrial Application, Hong Kong, vol.III, pp. 387 – 390, 2011

[5] W.-C. Hsu, S.-T. Liu, "Design and Implementation of CAN-USB Converter Based on ARM7 Serial Protocol API", International Symposium on Computer, Consumer and Control, Taichung, pp. 333 – 336, 2012

[6] M. Postolache, C. Spiridon, "XCPI – A Measurement and Calibration Software Tool for Networked and Embedded Control Systems", Proceedings of the 14th International Conference on Systems Theory and Control, Sinaia, pp. 397 – 402, 2010

[7] ST Microelectronics, Discovery kit with STM32F407VG MCU, User Manual (UM1472), 2017

[8] Axelson, USB Complete: The Developer's Guide, 5th Edition, LakeView Research LLC, Madison, 2015