Acta Universitatis Sapientiae
Electrical and Mechanical Engineering, 4 (2012) 58-69

Development of FPGA Based CAN Bus Error Generator System

Krisztian ENISZ[1], Denes FODOR[1], Balazs NEMETH[2], Ferenc SPEISER[1]

[1] Department of Automotive Mechatronics,
Faculty of Engineering,
University of Pannonia, Veszprem, Hungary,
e-mail: eniszk@almos.uni-pannon.hu
[2] Continental Automotive Hungary Ltd., Veszprem, Hungary

Manuscript received March 15, 2013; revised May 15, 2013

Abstract: This paper introduces the development and architecture of an FPGA based communication error generator system for CAN (Controller Area Network) networks. The created test-system is able to receive and interpret the frames on the network and capable to modify the user specified part of these messages on bit level in real-time. Beside of these modifications the environment has to be feasible to generate physical level error.

Keywords: automotive, CAN, FPGA, error generation

1. Introduction

In a modern vehicle there are more than 40 Electronic Control Units (ECUs) and there are countless sensors and actuators. The communication between these devices is implemented on different types of networks and protocols [1] [2]. In an ordinary car 2-3 different kinds of communication protocols are applied [3]. Most of the electronic control units including the safety critical systems are connected by Controller Area Network (CAN). These safety systems like the anti-lock braking system (ABS) and the electronic stability program (ESP) help to preserve the stability of the vehicle [4]; therefore it is indispensable to test these systems in every respect; including the validation and verification tests of communication.

On an average CAN bus more than 2000 signals and more than 200 frames are transmitted with even 1 Mbit/s baudrate. Hence it can be observed that even a short disturbance or error in the communication can cause large amount of data loss or corruption. The corrupted or missed data can determine the malfunction of safety critical system that can cause an accident if the ECU is not fault-tolerant.

A standard communication test generally includes the physical and logical test of the transceiver modules of the CAN nodes. The main point of these tests is to examine that the CAN nodes are able to properly create and process the CAN frames and signals and that they can handle the disturbances and errors. If there is no hardware error at CAN nodes, it is not so complicated to test the communication because it is enough to measure the transmission and reception time of the frames and compare the structure and contents of the frames with the expected data. However, the fault insertion is not so simple. The simulation and test environment has to create errors and disturbances in real-time (e.g.: wire break, short circuit, bit level errors) in a reproducible way.

2. Existing systems

There are a few special devices (e.g.: GEMAC (CAN-Bus Tester), IXXAT (CANcheck)) on the market, which make it possible to implement similar tests. These are mostly network testers and analyzers for physical diagnostic of CAN line and communication (e. g. bus state, bus load, monitoring); a few of them are capable to generate network and physical level errors. One of the most popular devices that solves these problems is produced by the Vector company.

The Vector CANstressDR modulates the state of the bus by software tunable electronic components. It is able to emulate bus line errors, disturb the CAN transceiver, create short circuit and wire break etc. This device is a good choice to facilitate the development of ECUs. With these tools it is easy to create a communication test system, but there are some disadvantages e.g. the price of these devices is relatively high and it is hard to adapt these “closed” systems to special tests.

That was the reason why the development of a new communication test system has been started. The creation of a new hardware and software environment is not time and cost efficient [5].

3. Concept

In the modern ECU development methods the cost efficient and thorough testing is very important in order to decrease the possibility of errors. One of the testing procedures is the Hardware In the Loop (HIL) test, which is widely used in different levels of the development to validate the functionalities. For the HIL tests the tested device are generally connected to a high performance real-time system. The main function of the HIL environment is to provide the signals which are necessary for the ECU to be able to operate and receive the data and signals from the tested device.

The modern multi-functional test environment has different kinds of analog and digital input and output ports and communication interfaces. It can be beneficial to use the existing interfaces to create a communication test system from general purpose test environment, in order to minimize the hardware investment. The other advantage of this concept is that it makes possible to customize the software for special tasks and that it and can be integrated into the different types of tests.

The elaborated test system is based on the PXI platform of National Instruments which is originally built for HIL testing of ECU's. The system is equipped with an x86 (x64) processor that is able to execute simulation steps with 1ms cycle time thanks to the Phar Lap real-time operating system.

The special high speed real-time controlling, data acquisition, signal processing tasks are executed by the FPGA based analogue and digital I/O modules. These modules are freely reconfigurable using the NI LabVIEW graphical programming environment and the FPGA is able to act as an individual processing unit. The system is also equipped with CAN, LIN, FlexRay and other automotive communication interfaces.

Since it was not the point to develop a generic device like the above mentioned Vector CANstressDR, it was possible to introduce several restrictions in the design phase of the communication test-system. The only requirement was to be able to freely modify the communication both on hardware and software level.

The error-generator environment should meet the following requirements:

- it should be sufficient to modify the messages going towards the tested device;
- the interaction should be invisible for the tested device;
- the message modification, the state change of the communication line should happen in near real-time;
- the collision of the messages before and after the modification is not allowed, the two messages should appear as two separate lines;
- it should be possible to modify any of the bits of the messages;
- it should be able to produce line-cut on any of the CAN lines.

For the error generation on the CAN bus it is necessary to connect the error generator unit to the bus. If the error generator unit is connected in parallel to the bus as a node of the network, then it is not possible to fully manipulate the state of the bus without a special hardware. It is not possible to overwrite the dominant state to recessive in this configuration, because of the structure of the physical layer [1]. Bearing in mind the requirements, a new architecture has been designed, which is different from the Vector system. While in this case the goal was the communication between the two devices, the solution was to insert the error generator between the devices as a special gateway (*Fig. 1* and *Fig. 2*).

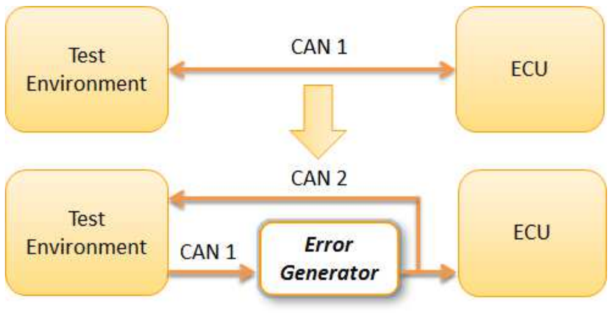

Figure 1: The main concept of the error generator system.

This architecture ensures that the original and the modified messages are available during the test and sending of the original message is not influenced by the error generator system. It is one of the main requirements in respect of the system to realize the message processing and error generation in near real-time. This can be managed by the special FPGA module of the system.

The maximum transmission speed of the CAN bus is 1 Mbit/s. This means that is 1 μsec to receive the actual bit, modify it according to the user demands, store the necessary data and the modifications of it, and transmit it to the tested device.

The computing performance of a general system with x86 (x64) architecture is not enough for the task; only a special, high performance microcontroller, DSP or FPGA is able to manage this complex problem.

4. Implementation

Fig. 2 shows the architecture of the error generator system. The controller of the system is an NI PXIe-8133 embedded controller with extension modules in an NI PXIe-1062Q 8-slot chassis. The NI PXI 8513/2 NI-XNET CAN interface module is responsible for the CAN communication, but the low level frame processing and the error generation is running on a Virtex-5 LX30 high-performance FPGA unit; integrated into a 7851R R series multifunctional reconfigurable I/O module.

Because of the application of the FPGA-based card for the fast processing and pre-processing (e.g.: filtering, signal-level interpretation) it is worth to use the own digital I/O interfaces of the card instead of using the analogue I/O interfaces with sampling-based signal-level detection. In this way these tasks are managed by the card. The digital I/O interfaces use standard TTL signal levels (0-5V), while the CAN applies differential voltage based physical layer. A CAN transceiver module (ADM3053EBZ) produced by Analog Devices was used to resolve this problem. This device is supposed to do the voltage-level transformation and the galvanic isolation.

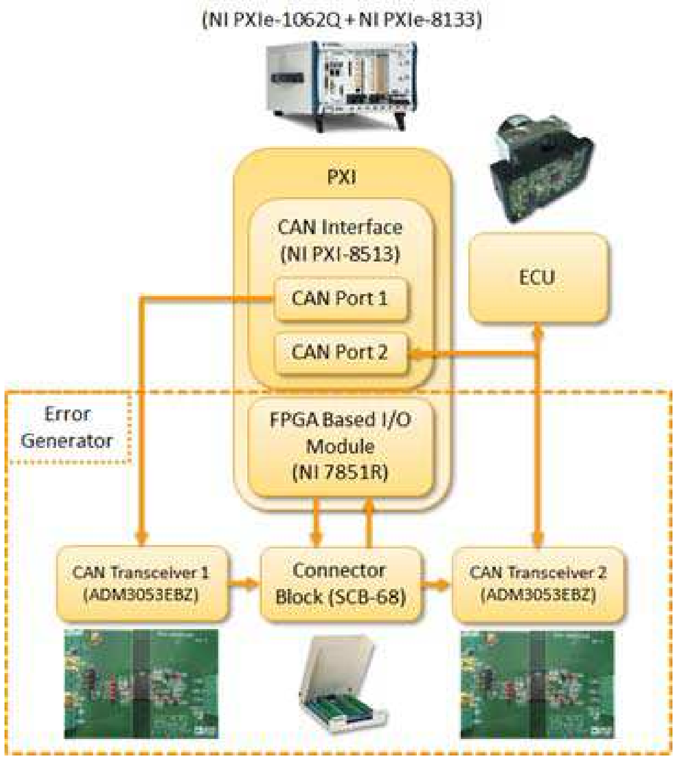
(NI PXIe-1062Q + NI PXIe-8133)
PXI
CAN Interface (NI PXI-8513)
CAN Port 1
CAN Port 2
ECU
Error Generator
FPGA Based I/O Module (NI 7851R)
CAN Transceiver 1 (ADM3053EBZ)
Connector Block (SCB-68)
CAN Transceiver 2 (ADM3053EBZ)

Figure 2: The structure of the error generator system.

At the early stage of development the error generator algorithm was embedded into a LabView FPGA project. The program structure is a kind of state machine that is composed of three main parts. After execution it waits, until the CAN bus comes into idle state, namely there is no communication between the devices on the network. The algorithm starts from this standby state and waits for a valid Start bit of a CAN frame. After the reception of the Start bit, the algorithm steps into the processing and modification state.

For every part of the CAN frame (*Fig. 3*) there are different sub-states (there are 10 different states for the standard frames and 3 additional states for the extended frames). In these sub-states the program receives and processes the incoming bits, modifies them if it is necessary and forwards the bit. When the reading of all of the bits of the actual frame-state is finished, the program switches to the next state. After the last state the program reinitializes the variables and returns to the waiting state at the beginning. There is a special sub-state, the acknowledgement that is necessary for the smooth communication. During the acknowledgement the sender device releases the bus (sets it to recessive state), and the receiver devices set the bus to dominant state for on bit time long if the frame was read successfully (in case of errorless communication).

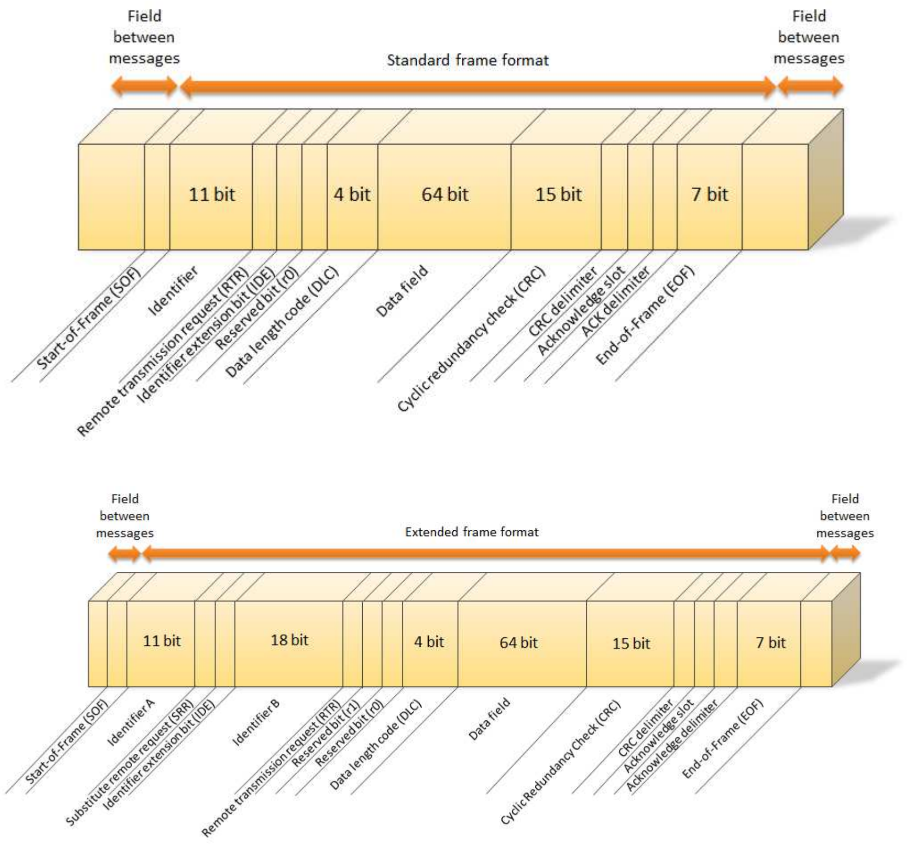

Figure 3: The structure of the standard and of the extended CAN frame.

The exact timing of the sampling point and of the start and end of program states is indispensable for the appropriate operation of the program. This can be managed by the FPGA with a resolution given by its clock period. The control program is complicated because of the applied synchronization process of the

CAN (bit-stuffing). If five bits arrive with identical value, then a bit must be inserted with an opposite value during the transmission. The program is able to identify these inserted bits and throws them away.

Furthermore, in order to meet the preliminary specifications, the program is able to generate physical error (line cut) by cutting both of the lines of the CAN bus separately with the help of a NI 9485 8 channel solid state relay module (placed into the system with the help of an NI 9151 extension chassis).

Fig. 4 below shows the simplified flow-diagram of the program.

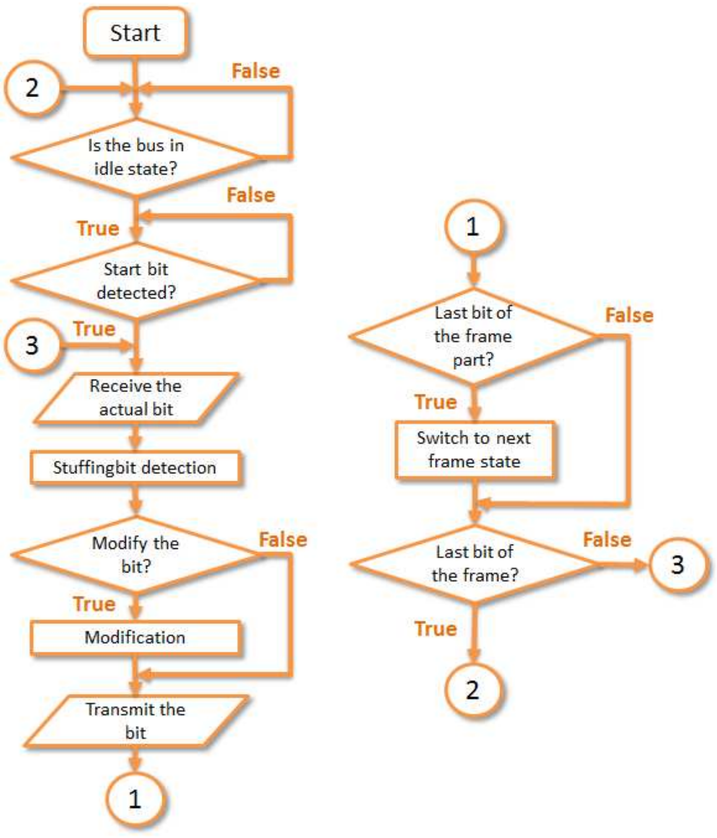

Figure 4: The flow-chart of the error generator program.

In the final development stage, after successful development and implementation of the basic algorithm, the FPGA program was integrated into a higher application layer using National Instruments’ VeriStand. The VeriStand is a software interface for real-time testing and simulation applications. VeriStand gives the possibility to make the frame modifications not just manually but automatically, controlled by a host PC with predefined stimulus-profiles from different test cases. *Fig. 5* shows the final system architecture.

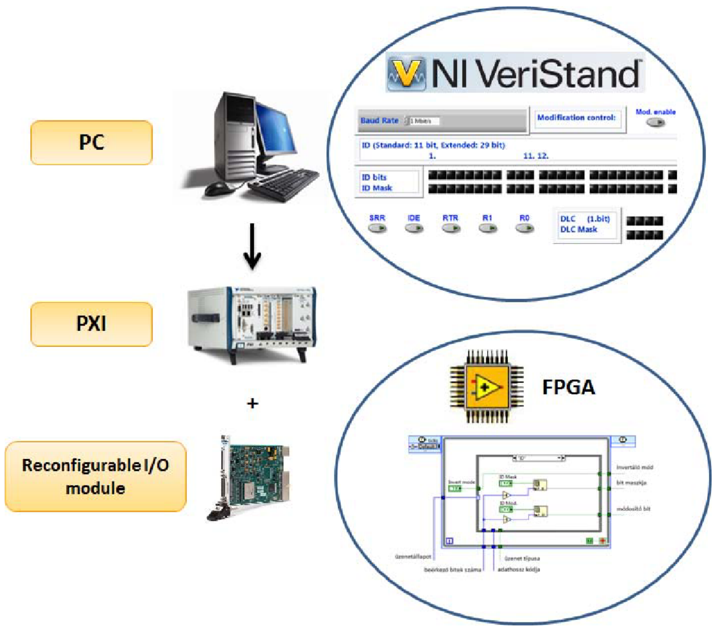

Figure 5: The structure of the system.

The error generator software runs on the reconfigurable FPGA module in the PXI that is controlled by an interface in a VeriStand project on the host PC. The NI VeriStand Real-Time engine implements the communication between the host PC and FPGA module in the PXI.

5. Measurements

One of the main requirements for the system was that it has to be able to modify the CAN frame in a near real-time manner. In order to verify this condition, measurements were made using the elaborated test-system during the development cycle. The goal of the measurements was to define the signal delay that arises from the normal operation of the error generator system. The reason of this delay is that processing and modification of the original frame take some time and the system is able to forward the message only with a small time delay. By minimizing of this latency the near real-time error generation can be achieved. For the CAN protocol the bit-timing is divided into specific sections and the possible place of sampling is also defined.

In order to determine the value of a bit, every CAN device on a network takes samples between the 50% and 80% of the bit-time [6]. It is therefore important to partly overlap the sampling ranges of the bidirectional devices for the shift in-between the signals. In this case the length of the time-shift should be smaller than the half of the bit-time; and the transfer of the bits should be near real time.

During the described measurement, the measurement system consisted of:

- the elaborated error generator system;
- IXXAT Automation GmbH USB-to-CAN Compact (USB-CAN interface);
- Tektronix MSO4054B oscilloscope.

The IXXAT USB-CAN external interface helped with the transmission of the messages intended to be modified. The original and the modified messages were compared by using an oscilloscope to validate the results of the modification and to measure the parameters of the signals.

Fig. 6 shows the modification of the original normal frame message with 71D(hex) identifier to 5D(hex) identifier at the transfer speed of 500 Kbit/s. In the picture the state of the receiver side of the CAN bus (original message) is displayed as yellow, the state of the tested device side CAN bus as cyan (modified message). The VeriStand based user interface is shown as well.

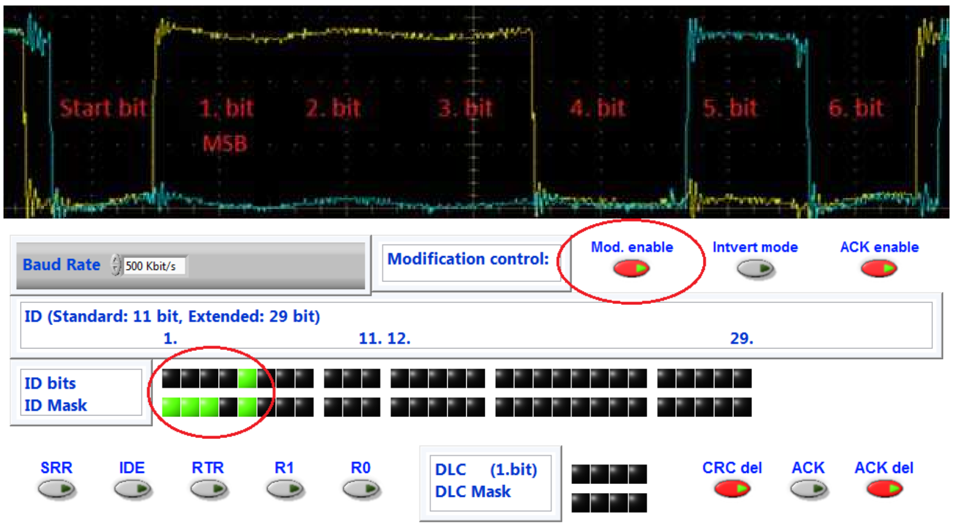

Figure 6: Modification of the message with 71D (hex) identifier.

Table 1 presents the average delays of five measurements resulted at different transfer speeds (the coefficient variation is always smaller than 3%).

Table 1: Signal delay as a function of transfer speed

Transfer speed [Kbit/s]	**Signal delay [ns]**	**Bit-time [ns]**	**(Signal delay/ Bit-time)·100 [%]**
1000	360	1000	36
800	380	1250	30.4
500	440	2000	22
250	526	4000	13.15
125	534	8000	6.675
50	516	20000	2.58
20	526	50000	1.052
10	532	100000	0.532

Based on the data of the table above, it appears that the system meets the requirements for the signal delay [7], while the signal delay is only 36% of the full bit time at 1 Mbit/s transfer speed. Furthermore *Fig. 7* and *Fig. 8* show the oscilloscope screenshots of the signal delays during the measurements at 125 and 500 Kbit/s transfer speeds. The CAN high and CAN low line of the transmitter device (IXXAT) are shown as yellow and blue. The high and low lines of the bus on the output side of the error generator system are shown as purple and blue.

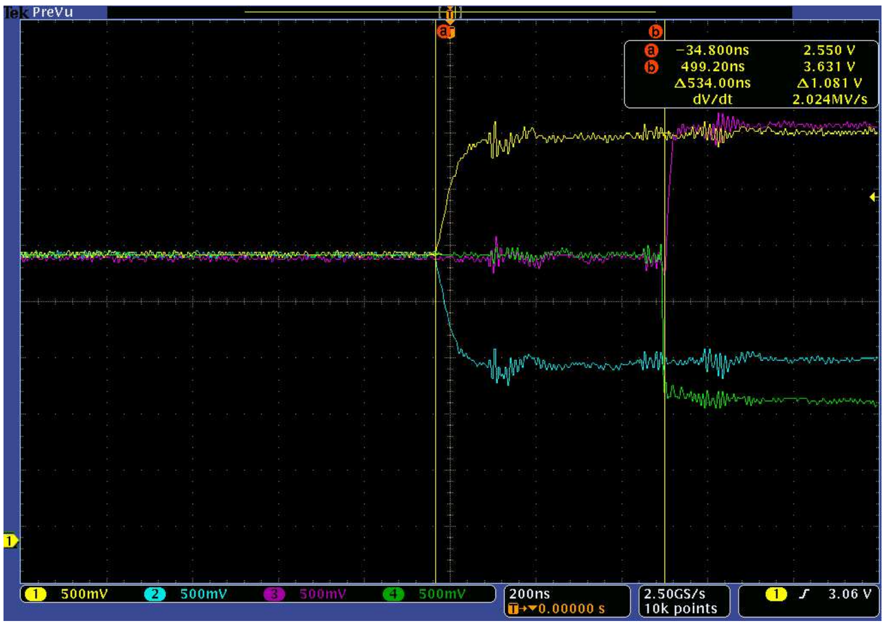

Figure 7: Signal delay at 125 Kbit/s transfer speed (yellow and cyan are the original, magenta and the green are the modified CAN-H and CAN-L lines).

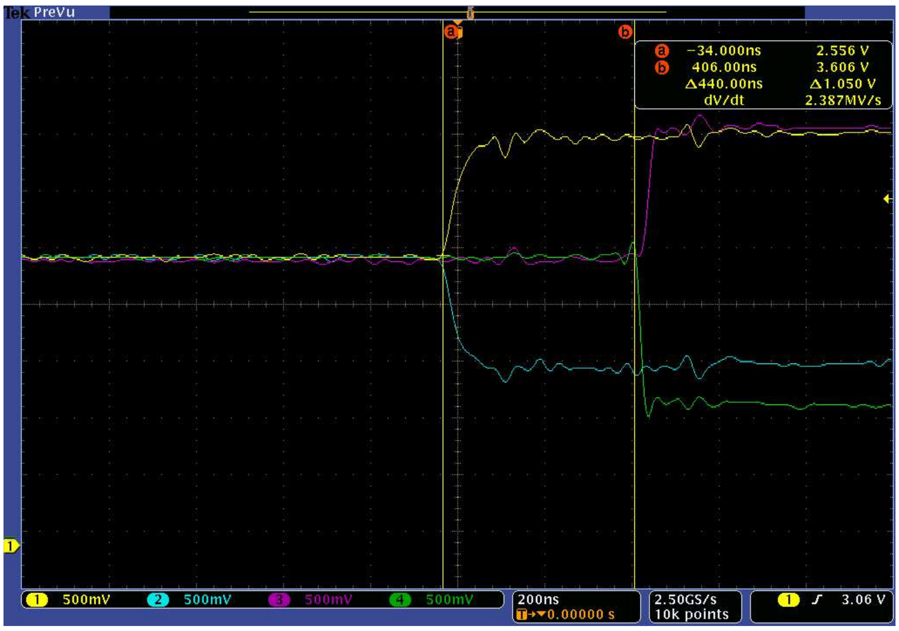

Figure 8: Signal delay at 125 Kbit/s transfer speed (yellow and the cyan are the original, magenta and green are the modified CAN-H and CAN-L lines).

3. Conclusion

During the implementation of the project it had been proven, that a general purpose simulator – thanks to its special features – only by making minimal hardware changes on it can, be suitable to replace a possibly expensive test device.

Furthermore the project came up with a system that is able to receive CAN messages with normal and extended frames in real-time. Any of the bits of the received frames can be inverted or set to a predefined value, while the system bus-speed can be set to the eight most commonly used transmission speeds from 10 Kbit/s up to 1 Mbit/s. According to the test results obtained using a GM ABS ECU, it can be stated that the elaborated system meets the specified requirements.

References

[1] CAN specification version 2.0. Robert Bosch GmbH, Stuttgart, Germany, 1991.

[2] Etschberger, K., “Controller Area Network”, in *IXXAT Press*, Weingarten, Germany, 2001.

[3] Nolte, T., Hansson, H., Bello, L. L., G., “Automotive communications-past, current and future”, in *IEEE Conference on Emerging Technologies and Factory Automation*, 2005.

[4] Johansson, K. H., Torngren, M., Nielsen, L., “Vehicle Applications of Controller Area Network”, in *Springer Control Engineering*, 2005, pp. 741-765.

[5] Novak, J., Fried, A., Vacek, M., “CAN Generator and Error Injector”, in *Proceedings of IEEE International Conference on Electronics, Circuits and Systems*, Dubrovnik, Croatia, September 2002, pp. 967-970.

[6] Philips Electronics N. V., “Determination of Bit Timing Parameters for the CAN Controller SJA 1000”, in *Application Note AN97046*, 1997.

[7] Novak, J., “New measurement method of sample point position in controller area network nodes”, in *Elsevier Measurement*, Vol. 41, Issue 3, April 2008, pp. 300-306