A SOFTWARE TOOL FOR VEHICLE CALIBRATION, DIAGNOSIS AND TEST VIA CONTROLLER AREA NETWORK

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

UTKU CİVELEK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2012

Approval of the thesis:

A SOFTWARE TOOL FOR VEHICLE CALIBRATION, DIAGNOSIS AND TEST VIA CONTROLLER AREA NETWORK

submitted by **UTKU CİVELEK** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** ________________

Prof. Dr. İsmet Erkmen
Head of Department, **Electrical and Electronics Engineering** ________________

Assoc. Prof. Dr. Ece Güran Schmidt
Supervisor, **Electrical and Electronics Engineering Department, METU** ________________

Examining Committee Members:

Prof. Dr. Semih Bilgen
Electrical and Electronics Engineering Dept., METU ________________

Assoc. Prof. Dr. Ece Güran Schmidt
Electrical and Electronics Engineering Dept., METU ________________

Assoc. Prof. Dr. Cüneyt Bazlamaçcı
Electrical and Electronics Engineering Dept., METU ________________

Assoc. Prof. Dr. Halit Oğuztüzün
Computer Engineering Dept., METU ________________

Utku Karakaya (M.S.)
R&D EE, TOFAŞ ________________

Date: ________________

ABSTRACT

A SOFTWARE TOOL FOR VEHICLE CALIBRATION, DIAGNOSIS AND TEST VIA CONTROLLER AREA NETWORK

Civelek, Utku
M.S., Department of Electrical and Electronics Engineering
Supervisor : Assoc. Prof. Dr. Ece Güran Schmidt

September 2012, 65 pages

Controller Area Networks (CAN's) in vehicles need highly sophisticated software tools to be designed and tested in development and production phases. These tools consume a lot of computer resources and usually have complex user interfaces. Therefore, they are not feasible for vehicle service stations where low-performance computers are used and the workers not very familiar with software are employed. In this thesis, we develop a measurement, calibration, test and diagnosis program -diaCAN- that is suitable for service stations. diaCAN can transmit and receive messages over 3 CAN bus channels. It can display and plot the data received from the bus, import network message and Electronic Control Unit (ECU) configurations, and record bus traffic with standard file formats. Moreover, diaCAN can calibrate ECU values, acquire fault records and test vehicle components with CAN Calibration Protocol functions. All of these capabilities are verified and evaluated on a test bed with real CAN bus and ECUs.

Keywords: Controller Area Network(CAN) bus, Electronic Control Unit (ECU), vehicle diagnosis and calibration

ÖZ

ARAÇ KALİBRASYONU, HATA TEŞHİSİ VE TESTİNİ DENETLEYİCİ ALAN AĞI (CAN) ÜZERİNDEN YAPAN BİR YAZILIM

Civelek, Utku
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Ece Güran Schmidt

Eylül 2012, 65 sayfa

Taşıtlarda kullanılan Denetleyici Alan Ağlarının (CAN'lerin) hem tasarımı hem de geliştirme ve üretim aşamalarındaki testleri son derece gelişmiş yazılımlar gerektirmektedir. Bu yazılımlar sistem kaynaklarının büyük bir bölümünü tüketmekte ve kullanıcı arayüzleri de genelde karışık olmaktadır. Bu yüzden, düşük performanslı bilgisayarların kullanıldığı, yazılımlara pek aşina olmayan personelin çalıştığı servis istasyonları için kullanışsız kalmaktadırlar. Bu tez kapsamında, servis istasyonları kullanımına uygun bir ölçüm, kalibrasyon, test ve hata teşhis programı -diaCAN- geliştirilmiştir. diaCAN 3 CAN veri yolu kanalından mesaj alıp gönderebilir, elde edilen bilgileri ekranda gösterip grafiklerini çizebilir. Standart dosya biçimlerini kullanarak ağ mesajlarının ve Elektronik Kontrol Ünitelerinin (ECU'ların) yapılandırmalarını yükleyebilir; veri yolu trafiğinin kaydını tutabilir. Ayrıca CAN Kalibrasyon Protokolü (CCP) fonksiyonları aracılığıyla ECU değerlerini kalibre edebilir, hata kayıtlarını edinebilir ve araç bileşenlerini test edebilir. diaCAN'in tüm bu özellikleri gerçek CAN veri yolu ve ECU'lar içeren bir test düzeneği ile değerlendirilip doğrulanmıştır.

Anahtar Kelimeler: Denetleyici Alan Ağı veri yolu, Elektronik Kontrol Ünitesi (ECU), araç hata teşhisi ve kalibrasyonu

To my parents and my friends in METU Turkish Folklore Club

ACKNOWLEDGMENTS

First of all, I am indebted to my thesis supervisor Assoc. Prof. Dr. Şenan Ece Güran Schmidt for guiding me throughout my thesis development.

I would like to thank Utku Karakaya, project leader in TOFAŞ, for the pleasant and efficient communication he carried out for more than one year. Participating in an electrical vehicle development process was a great pleasure for me.

I am owed to Enes Ünal due to his assistance especially at the beginning of the software development process. My other friends in METU Turkish Folklore Club also deserve an appreciation since they've made my campus life delighted and peaceful.

Finally, I would like to thank my parents for their love and support. They have made my academic education the priority of their lives for 20 years and have the biggest share in my position.

TABLE OF CONTENTS

ABSTRACT iv

ÖZ v

ACKNOWLEDGMENTS viii

TABLE OF CONTENTS ix

LIST OF TABLES xii

LIST OF FIGURES xiii

CHAPTERS

1 INTRODUCTION 1

2 ELECTRONIC CONTROL UNITS AND CONTROLLER AREA NETWORKS IN VEHICLES 3

2.1 Controller Area Network (CAN) 4

2.1.1 CAN Messages 4

2.1.2 CAN Signals 5

2.2 Calibration 5

2.3 File Formats 8

2.3.1 .dbc Database File 8

2.3.1.1 Message Information 9

2.3.1.2 Signal Information 9

2.3.2 .asc Log File 11

2.3.3 .a2l ECU Configuration File 12

3 EXPERIMENTAL EQUIPMENT 14

3.1 Kvaser USBcan Professional 14

3.2 ECU 14

3.3 Kvaser CanKing 15

4 DiaCAN SOFTWARE DEVELOPMENT 16
4.1 Development Environment 18
4.2 Overview 18
4.2.1 acilirCAN 18
4.2.2 Base Window 19
4.2.3 Other Windows 20
4.3 Detailed Structure 26
4.3.1 Base Window 26
4.3.2 Main Thread 27
4.3.3 Thread Array 28
4.3.4 Registers 29
4.3.5 Data Tables 31
4.3.6 Timers 32
4.3.7 Counters 33
4.3.8 Flags 34
4.3.9 Global Variables 35
4.3.9.1 Message Variables 35
4.3.9.2 Control Variables 36
4.3.9.3 Fault Variables 37
4.3.10 CCP Variables 37
4.3.11 Base Window Items’ Functions 37
4.3.11.1 Top Items 37
4.3.11.2 Messages and Signals Tab 39
4.3.11.3 Log Tab 39
4.3.11.4 CCP Tab 40
4.3.11.5 Settings Tab 42
4.3.12 Base Functions 43
4.3.13 grafiCAN 45
4.3.14 mesajCAN 49
4.3.15 sinyalCAN 50

4.3.16 bilgiCAN 51

4.3.17 Info Screen 51

4.4 Summary of diaCAN Operation 52

4.5 Sytem Requirements & Development Workload 53

4.6 Licensing Software by Device Pairing 53

5 TEST RESULTS 54

6 PREVIOUS WORK ON CAN/ECU SOFTWARE 57

7 CONCLUSION 62

REFERENCES 64

LIST OF TABLES

TABLES

Table 5.1 diaCAN Test Results 55

Table 6.1 CAN-Software Modules & Prices 60

Table 6.2 CAN-Software Capabilities 61

LIST OF FIGURES

FIGURES

Figure 2.1 CAN Frame Format 5

Figure 2.2 Position of CCP protocol in OSI model 6

Figure 2.3 Main types of CCP messages 7

Figure 3.1 Kvaser CanKing interface 15

Figure 4.1 Development Environment 18

Figure 4.2 diaCAN windows 19

Figure 4.3 diaCAN base window 21

Figure 4.4 diaCAN base window 22

Figure 4.5 diaCAN base window 23

Figure 4.6 diaCAN base window 24

Figure 4.7 diaCAN base window 25

Figure 4.8 diaCAN base window components 26

Figure 4.9 Registers 30

Figure 4.10 Data Tables 32

Figure 4.11 Use of Global Step 34

Figure 4.12 Flags 34

Figure 4.13 Global Variables 36

Figure 4.14 Items at the top of diaCAN base window 38

Figure 4.15 Menu Strips 38

Figure 4.16 Items in the first tab of diaCAN base window 39

Figure 4.17 Items in the second tab of diaCAN base window 40

Figure 4.18 Items in the third tab of diaCAN base window 40

Figure 4.19 Items in the fourth tab of diaCAN base window 43

Figure 4.20 *grafiCAN* window 48

Figure 4.21 *mesajCAN* window 49

Figure 4.22 *sinyalCAN* window 50

Figure 4.23 *bilgiCAN* window 51

Figure 4.24 Info Screen 51

Figure 4.25 diaCAN items and functions' interaction 52

Figure 5.1 Jitter Observed in Periodic Messages 56

CHAPTER 1

INTRODUCTION

Controller Area Networks (CAN's) in vehicles get more complicated with the increasing number and capabilities of in-vehicle electronic components.

At the factory side, engineers use highly advanced proprietary software solutions[1][2][3][4] to design, manage and test CAN traffic and nodes. These programs require PC's with high CPU and RAM capacities, users with advanced computer skills and English knowledge. Thus, they do not suit to vehicle service stations, where no design process is performed. In these stations simple, compact and user-friendly software is preferable. There exist many free and inexpensive software solutions to be used in service stations for sending, receiving and logging CAN messages, but they do not offer features such as graphical representation of multiple signals, tracking arithmetic products of measured signals, automated ECU diagnosis, calibration and test with CAN Calibration Protocol.

Using more than one program may be another choice. There are programs specified on one or two CAN methods only. Concurrent use of them enables all service operation. However efficiency on the utilization of CAN bus gets worse since every program needs to track all messages on its own. Furthermore licensing cost gets higher and maintenance-update process gets harder.

In this thesis, a simple measurement, calibration, test and diagnostics program -diaCAN- is developed as a part of a university-industry cooperation project. Since this program will be used in service stations, resource demand of the program is kept low without risking data acquisition. A plain user interface is designed for the program; considering different levels of computer skills users may have. diaCAN works with standard ECU hardware different than previous work proposed in [5] and [6] which require hardware modifications. diaCAN is tested with real ECU signals and our test results demonstrate successful and efficient use of diaCAN in real environment.

The remainder of this thesis is organized as follows. Chapter 2 includes background information about automobile networks, controller area network (CAN), calibration of Electronic Control Units (ECUs) and file formats.

In Chapter 3, hardware and software tools used in development and experiment process of diaCAN are introduced.

Chapter 4 is dedicated to structure of diaCAN. Windows, threads, registers, data tables, timers, counters, flags, variables and functions of the program are explained in details and visualized with screen prints and schemas.

Chapter 5 includes results of software performance tests conducted in computers with different configurations

In Chapter 6, similar and more advanced software tools are analyzed. Diagnostics, calibration, and ECU development features of them are examined as well as their user interfaces, system requirements and prices.

Chapter 7 concludes the thesis with discussion of test results and possible enhancements of diaCAN.

CHAPTER 2

ELECTRONIC CONTROL UNITS AND CONTROLLER AREA NETWORKS IN VEHICLES

In this chapter, we present relevant background information to clarify technical aspects of diaCAN. First, ECU's and their roles in vehicles are explained. Then the networks for ECU communication in vehicles are described. Afterwards Controller Area Network (CAN) is introduced with its basic operation principles, calibration techniques and file formats.

Electronic control units (ECUs) in vehicles are used for implementing functions such as:

- driver assistance with acceleration, braking and steering functions such as the anti slip regulator (ASR), antilock braking system (ABS), and electronic brake distributor (EBR)
- easy control of user-modifiable body components such as air conditioner, lights, mirrors, windows, and wipers
- data transfer in vehicle entertainment and communication systems such as radio, MP3 player, and Global Positioning System (GPS) devices.[7]

The first developed ECUs were implementing a single function, whereas tens of functions or data values are handled by a single ECU, today. These units include sensors, actuators, RAM, ROM and flash memory as well as a microcontroller to perform its operations.

Until 1980s, the in-vehicle communication among electronic devices was achieved using point-to-point wiring [8]. With the expanding number of electronic components within the vehicle, wiring became a volume, reliability, weight (fuel consumption) and test problem.

Vehicle networking was proposed as the solution to this problem by decreasing number of wires and easing control and monitoring of the system. A bunch of different automotive networks were developed by several companies. Controller Area Network (CAN) became the most successful standard among them.

2.1 Controller Area Network (CAN)

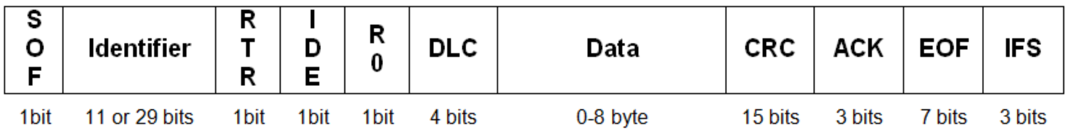
Controller Area Network is a serial bus developed for automotive applications in 1980s by Robert Bosch GmbH [9]. It was standardized in 1993 as ISO 11898 [10] and revised last in 2003.

In a Controller Area Network, transmission is controlled with the help of unique identifiers enclosed in CAN messages. These identifiers (CAN IDs) state priority and content of the messages. Since sender addresses are not used for transmission, adding new nodes does not require any modifications. Another benefit of the system is multicasting: a CAN message can be received and used by many receivers at the same time.

Carrier Sense Multiple Access/Collision Resolution (CSMA/CR) protocol is used for CAN buses. Accordingly, the nodes listen to the channel and do not start transmission when the channel is busy. Any node can start transmission when the channel is idle. If multiple stations simultaneously start then there is collision and the message with the smallest CAN ID (the highest priority) continues to be sent. This is achieved by bit-wise arbitration where dominant bit (0) overwrites the recessive bit (1). Other colliding nodes with lower priorities stop their transmission and listen the channel until it becomes idle again.

2.1.1 CAN Messages

There are two message frame formats in CAN protocol: Messages with 11 bit identifiers and messages with 29 bit extended identifiers.



SOF	Identifier	RTR	IDE	R0	DLC	Data	CRC	ACK	EOF	IFS
1bit	11 or 29 bits	1bit	1bit	1bit	4 bits	0-8 byte	15 bits	3 bits	7 bits	3 bits

Figure 2.1: CAN Frame Format

There are four types of frames for CAN Messages: [11]

- Data frames transferring data
- Remote frames used to demand transmission of the data frame with the same CAN ID
- Error frames indicating any error on the bus
- Overload frames used to set extended delay the preceding and the succeeding frames

An inter-frame space separates consecutive frames from each other.

2.1.2 CAN Signals

The signals that are transmitted by the ECU's constitute the data values transmitted on the CAN bus. These data can be encoded in bit sequences as short as 1-bit. Therefore several such data values are packed into one single CAN message to ease transmission and a CAN message can include tens of embedded signals.

2.2 Calibration

It is possible to read and change ECU parameter values over CAN bus by calibration protocols such as CAN Calibration Protocol(CCP), Keyword Protocol 2000 (KWP2000), Unified Diagnostic Services (UDS), On-Board Diagnostics (OBD) and Universal Measurement and Calibration Protocol (XCP). We focus on the CCP in the scope of this thesis.

CAN Calibration Protocol [12] is developed by Robert Bosch GmbH and Intel Corporation together. It is a top (7th) layer protocol according to the OSI model (Figure 2.2), used to acquire and modify controllers' data in both automotive networks and industrial control systems. Therefore programming of ECU's and tests of both ECU's and vehicle components are carried out via CCP.

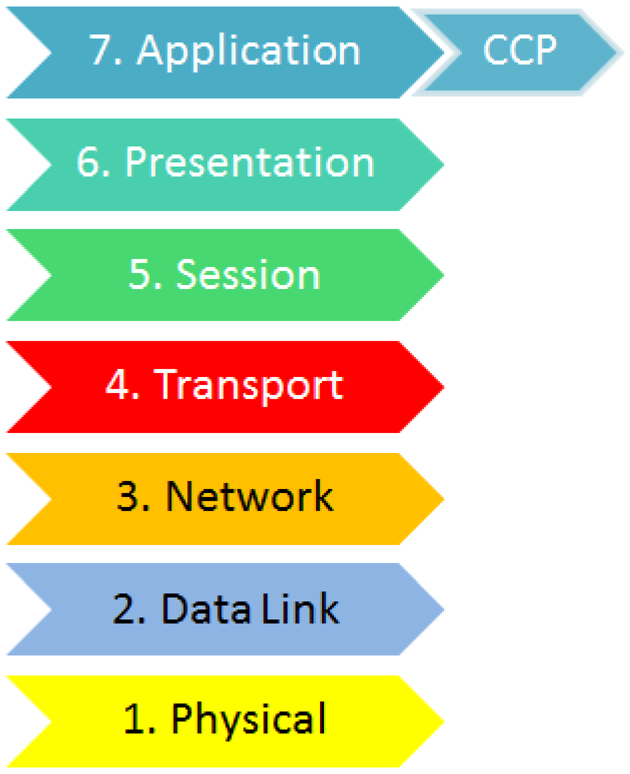


Figure 2.2: Position of CCP protocol in OSI model

CCP uses master-slave configuration where a master component such as a PC with calibration or measurement software starts the communication with ECUs which are slave components. No synchronization is required since data transfer is carried out step by step. The master sends initialization command in a single CAN message first, to set the logical connection. Data acquisition commands are transmitted afterwards. Target slave node is expected to reply all commands with a proper command return code or error code. Throughout the communication, only two CAN IDs are used (one for master, one for slave).

There exist 2 main types of message objects: Command Receive Object (CRO) sent from the master device and Data Transmission Object (DTO) sent from slave(s). A DTO message can be a Command Return Message (CRM), an Event Message or a Data Acquisition Message (DAQ).

The CRO consists of bytes representing type and related data of a command (Figure 2.3). The first byte is a hexademical number corresponding to command type. The second byte is the command counter representing sequence number of the command. Response message of ECU includes the same command counter value to ease the tracking of two-way communication.

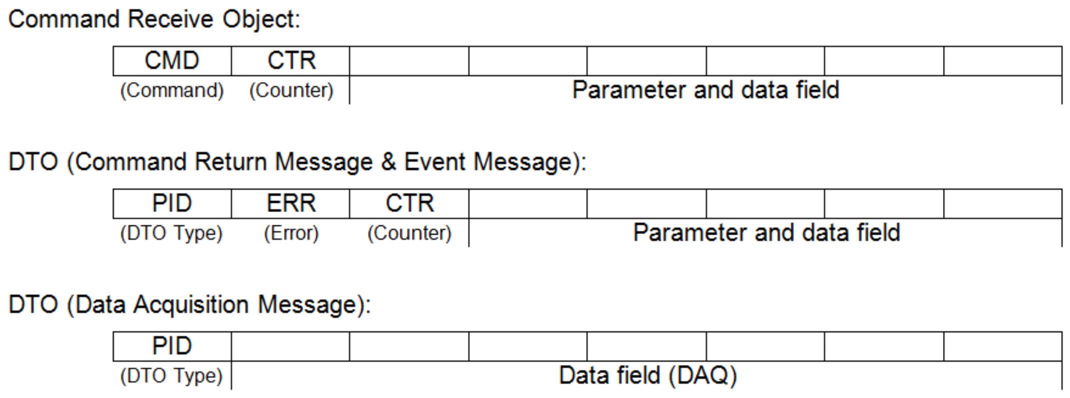


Figure 2.3: Main types of CCP messages

The CRM is the response message of ECU to any CRO. In addition to acknowledgment information, it can include ECU data if demanded. The Event Message reports changes that occurred in ECU status due to different events. Event Messages are commonly used when an error is observed after the transmission of the latest CRO.

As shown in Figure 2.3, message structure of the CRM and the Event Message are similar. The first byte is the Packet Identifier (PID), which is 0xFF for the CRM and 0xFE for the Event Message. The second byte is the error code and the third code is the command counter set to the value of last received CRO. Other 5 bytes are allocated to data related to response.

The DAQ transmits the measurement data to the master tool. For generation of this kind of messages, the master should initiate the logical connection by sending the requested data information. ECU indicates which data is event driven and which one is sampled periodically, in its response. Afterwards, measured values are transmitted to the master without the need of re-requests.

2.3 File Formats

Standardized file formats are used to save the configurations of ECUs and CAN buses. The logs of CAN message traffic are also stored in a standardized file. Next we present such three relevant file formats: DBC, ASC, and A2L.

2.3.1 .dbc Database File

The DBC file [14] describes the communication of a single CAN network. This information is sufficient to monitor and analyze the network. Furthermore ID-Data mappings of CAN messages and location of signals (which messages contain defined signals) are included in DBC files. Therefore messages and signals can be tracked and their replicas can be generated. Functional operations of ECUs are not defined by the DBC file.

We next describe the contents of the DBC files. The reserved keywords are listed below:

- BU_ : Network Node
- BO_ : CAN Message
- SG_ : CAN Signal
- EV_ : Environment Variable

DBC files include the following sections:

- Bit_timing (This section is normally empty.)
- Nodes (This section defines the network nodes.)
- Messages (This section defines the messages and the signals.)

In the scope of this thesis, only "Messages" section is relevant. This section defines the names of all frames, their properties and the signals transferred on the frames.

2.3.1.1 Message Information

The CAN-ID of each message has to be unique within the DBC file. If the most significant bit of the CAN-ID is set, the ID is an extended CAN ID. The extended CAN-ID can be determined by masking out the most significant bit with the mask `0xCFFFFFFF`.

```
message=BO_ message_id message_name':'message_size transmitter{signal};
```

Names of messages must be unique within the set of messages. The `message_size` specifies the size of the message in bytes. The transmitter name specifies the name of the node transmitting the message. The sender name has to be defined in the set of node names in the node section.

The transmitter name specifies the name of the node transmitting the message. The sender name has to be defined in the set of node names in the node section.

2.3.1.2 Signal Information

Signals are defined as follows:

```
signal = 'SG_' signal_name multiplexer_indicator ':'  start_bit
'|' signal_size '@' byte_order value_type '(' factor ',' offset ')'
'[' minimum '|' maximum ']' unit receiver ',' receiver ;
```

The names defined here have to be unique for the signals of a single message:

```
    signal_name = C_identifier ;
```

The `multiplexer` indicator defines whether the signal is a normal signal, a multiplexer switch for multiplexed signals, or a multiplexed signal. A 'M' (uppercase) character defines the signal as the multiplexer switch. Only one signal within a single message can be the multiplexer switch. A 'm' (lowercase) character followed by an unsigned integer defines the signal as being multiplexed by the multiplexer switch. The multiplexed signal is transferred in the message if the switch value of the multiplexer signal is equal to its `multiplexer_switch_value`.

```
    multiplexer_indicator = ' ' | 'M' | m multiplexer_switch_value ;
```

The `start_bit` value specifies the position of the signal within the data field of the frame. For signals with byte order Intel (little endian) the position of the least significant bit is given. For signals with byte order Motorola (big endian) the position of the most significant bit is given. The bits are counted in a sawtooth manner. The start bit has to be in the range of 0 to (8 * `message_size - 1`).

```
start_bit = unsigned_integer ;
```

The `signal_size` specifies the size of the signal in bits.

```
signal_size = unsigned_integer ;
```

The `byte_format` is 0 if the signal's byte order is Intel (little endian) or 1 if the byte order is Motorola (big endian).

```
byte_order = '0' | '1' ; (0=little endian, 1=big endian)
```

The `value_type` defines the signal as being of type unsigned (-) or signed (-).

```
value_type = '+' | '-' ; (+=unsigned, -=signed)
```

The `factor` and `offset` define the linear conversion rule to convert the signals raw value into the signal's physical value and vice versa:

```
factor = double ;

offset = double ;

physical_value = raw_value  factor + offset

raw_value = (physical_value - offset) / factor
```

As can be seen in the conversion rule formulas the `factor` must not be 0.

The `minimum` and `maximum` define the range of valid physical values of the signal.

```
minimum = double ;

maximum = double ;
```

`Unit` of the signal is a string of characters.

```
    unit = char_string ;
```

The `receiver` name specifies the receiver of the signal. The `receiver` name has to be defined in the set of node names in the node section.

```
    receiver = node_name ;
```

Signals with value types ’float’ and ’double’ have additional entries in `signal_valtype_list` section.

```
    signal_extended_value_type_list = 'SIG_VALTYPE_' message_id
signal_name
    signal_extended_value_type ';' ;

    signal_extended_value_type = '0' | '1' | '2' | '3' ; (0=signed or
unsigned integer, 1=32-bit IEEE-float, 2=64-bit IEEE-double)
```

2.3.2 .asc Log File

.asc is the extension of log file in ASCII format. It is used to record received CAN messages and their attributes. These records are used for offline analysis.There is no standard for data format of this file, therefore the format used by the popular Vector software tools is selected. A sample file is given below:

```
date Thu Feb 3 03:01:29 pm 2011
base hex timestamps absolute
internal events logged
Begin Triggerblock Thu Feb 3 03:01:29 pm 2011
0.000000 Start of measurement
0.000886 2 442 Rx d 8 40 FE 9C 3A FF EF FE 00
0.005984 2 260 Rx d 8 0F A0 3A 98 17 70 17 70
0.006720 2 262 Rx d 8 0F A0 00 00 82 00 00 00
0.006922 2 400 Rx d 4 01 82 00 00
0.012066 2 410 Rx d 8 08 0F FE 95 9C 3A 0F FE
.........
```

```
End TriggerBlock
```

Texts other than date and message values between "0.000000 Start of measurement" and "End TriggerBlock" are fixed for all .asc files.

2.3.3 .a2l ECU Configuration File

A2L (ASAP2) description files include information for data objects in the ECUs such as parameters, real and virtual measurement variables and variant dependencies. Memory addresses, saving structures, data types and arithmetic operations needed for converting them into physical units are included in this type of files. Moreover, ECU-computer communication parameters are also recorded in A2L files. [15]

Two examples of parameter address save in .a2l files are given below:

```
/begin CHARACTERISTIC
    /* Name */ C_MaxSpeed
    /* LongIdentifier */ ""
    /* Type */ VALUE
    /* Address */ 0x80000064
    /* Deposit */ RecordLayout_1
    /* MaxDiff */ 0
    /* Conversion */ CompuMethod_1
    /* LowerLimit */ 0
    /* UpperLimit */ 250
/end CHARACTERISTIC
```

```
/begin MEASUREMENT
    /* Name */ FaultMarquees.FaultMarquees.Current_Active
    /* LongIdentifier */ "Displays a rolling marquee for active faults."
    /* Datatype */ UBYTE
    /* Conversion */ CompuMethod_63
```

```
        /* Resolution */ 1
        /* Accuracy */ 0
        /* LowerLimit */ 0
        /* UpperLimit */ 255
        ECU_ADDRESS 0x400001F1
/end MEASUREMENT
```

CHAPTER 3

EXPERIMENTAL EQUIPMENT

In development and test periods of the software tool for ECU calibration, diagnosis and test over CAN bus, a CAN traffic generator software tool and two hardware tools are used. Kvaser USBcan Professional is the device used in PC-CAN connection. Kvaser CanKing software tool is used in traffic generation and an ECU is connected to Kvaser USBcan Professional for testing CCP functions.

3.1 Kvaser USBcan Professional

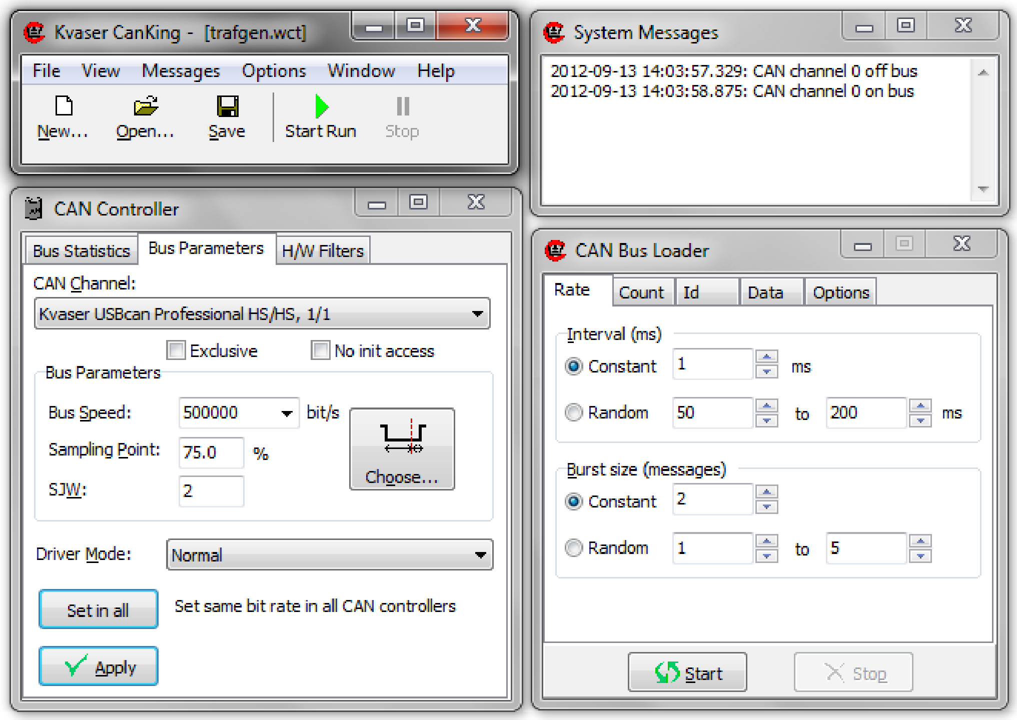
Kvaser USBcan Professional [32] is a two channel USB interface for the CAN bus enabling easy connection of different interfaces to a computer. It has a USB connector and two 9-pin D-SUB connectors. The device can handle up to 40,000 messages and buffer 100 incoming messages and around 50 outgoing messages per second. It supports both 11-bit (CAN 2.0A) and 29-bit (CAN 2.0B active) identifiers. CAN Messages are time-stamped and synchronized with a precision of 2 microseconds.

3.2 ECU

Since ECU programming is out of the scope of the project, an ECU & A2L file pair is provided by TOFAŞ for test purposes. This ECU is connected to 9-pin D-SUB connectors of Kvaser USBcan Professional. After importing the A2L file to diaCAN, ECU values are calibrated and read. Calibration is also used for modifying virtual fault generation values on ECU. Afterwards, these faults are displayed and cleared via CCP functions of diaCAN.

3.3 Kvaser CanKing

Kvaser CanKing [33] is a free CAN bus monitor and traffic generator tool. It can generate bursts of CAN messages on both real and virtual channels; therefore it is used in simulation and test of diaCAN.


Kvaser CanKing - [trafgen.wct]
File View Messages Options Window Help
New... Open... Save Start Run Stop
System Messages
CAN Controller
Bus Statistics Bus Parameters H/W Filters
CAN Bus Loader
Rate Count Id Data Options
Choose...
Set in all
Apply
Start Stop

2012-09-13 14:03:57.329: CAN channel 0 off bus
2012-09-13 14:03:58.875: CAN channel 0 on bus

CAN Channel: Kvaser USBcan Professional HS/HS, 1/1
Exclusive ☐ No init access ☐
Bus Parameters
Bus Speed: 500000 bit/s
Sampling Point: 75.0 %
SJW: 2
Driver Mode: Normal
Set same bit rate in all CAN controllers

Interval (ms)
● Constant 1 ms
○ Random 50 to 200 ms
Burst size (messages)
● Constant 2
○ Random 1 to 5

Figure 3.1: Kvaser CanKing interface

CHAPTER 4

DiaCAN SOFTWARE DEVELOPMENT

The scope of this thesis is the development and the evaluation of diaCAN which is a software tool for ECU calibration, diagnosis and test over CAN bus. It runs on a PC to receive/transmit CAN messages from/to CAN bus via a USB-CAN interface card. It is developed on .NET platform with C# programming language and compiled with Microsoft Visual Studio 2010.

diaCAN can transmit and receive CAN messages over 3 CAN bus channels. The user can change message attributes, signal values, and channel settings. He/she can also track signal values and sketch their graphs. Moreover, ECU values can be calibrated, fault records can be acquired and vehicle components may be tested via a user-friendly interface which handles CCP communication.

Main capabilities of diaCAN are as follows:

- Operating on 3 CAN bus channels (2 real channels and 1 real-or-virtual channel)
- Importing .dbc files as database
- Transmitting periodic and non-periodic user-modifiable CAN messages
- Displaying user-modifiable signals of CAN messages
- Tracking up-to-date values of selected signals (data)
- Drawing value-vs.-time graphs of selected signals
- Displaying last values of CAN messages updating tracked signals
- Displaying CAN message traffic on CAN bus

- Displaying and recording log (.asc) files
- Importing ECU configuration with .a2l files
- Showing former & current faults defined by ECUs and clearing fault records via CCP
- Calibration of ECU parameters via CCP
- Manage and monitor tests by modifying ECU parameters via CCP
- Operating with pre-licensed devices only

These features are basically controlled by the components of "Base Window". It handles data storage, program control and input/output operations. Although most of them are realized by immediate or timer-controlled functions, CCP operations are carried out a 3-phased diaCAN-specific CCP control process. This process is initiated when user enters a new value to a textbox or change the position of a trackbar on CCP tab of "Base Window". All demands from ECU are saved to a queue and processed one-by-one. Phases of this two-way asynchronous transmission are as follows:

- Phase 1: Transmission of the proper CRO message corresponding to user entry at the head of CCP queue
- Phase 2: Waiting for the response of ECU for the last send CRO message
- Phase 3: Processing received response and clearing temporary data used in these 3 phases

.NET structures used in these phases are explained in following sections of this chapter.

4.1 Development Environment

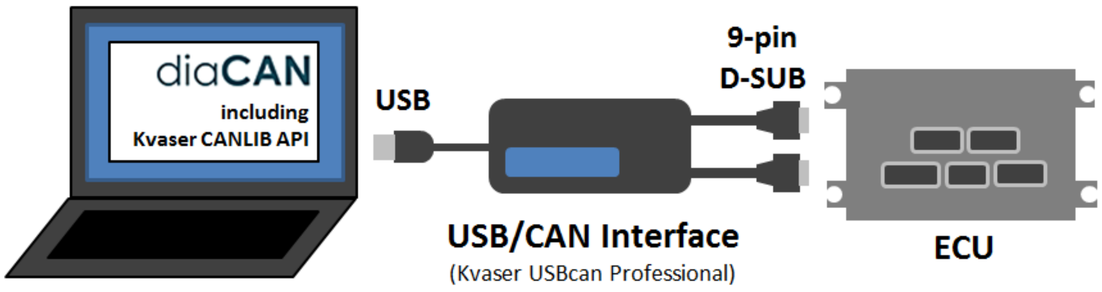


Figure 4.1: Development Environment

diaCAN software is developed by using Kvaser CANLIB API[35]. This API provides methods for obtaining CAN bus and device information, opening and closing CAN channels, getting and setting channel parameters, receiving and transmitting CAN messages. Since these methods are designed to work on Kvaser hardware components, diaCAN needs Kvaser-compatible USB-CAN interface devices to operate. These devices are plugged to USB and can be used immediately after driver installation.

4.2 Overview

diaCAN is a Windows form application with a pre-loading window called *acilirCAN* checking licensed hardware tools, followed by a main window (diaCAN base window) and subjacent *mesajCAN*, *sinyalCAN*, *bilgiCAN*, *grafiCAN* and program information window (Figure 4.2).

4.2.1 acilirCAN

On every start-up of diaCAN, connected devices to the PC are checked. If one of the pre-licensed USB-CAN interface devices is available, the software opens main window normally. If not, a pop-up window appears and asks user to connect a paired device. Pairing is established via manually embedding UPC (EAN) number of the card to the source code.

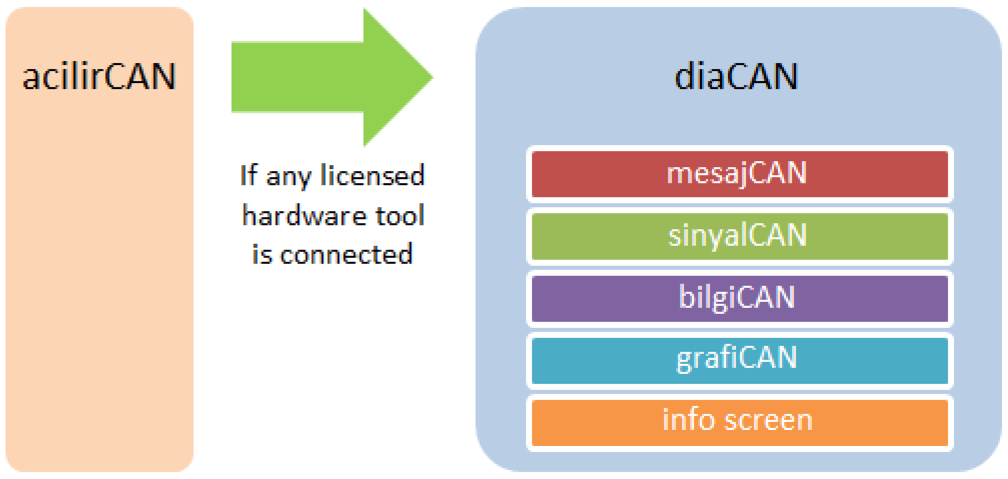


Figure 4.2: diaCAN windows

4.2.2 Base Window

On "Messages & Signals" tab (Figure 4.3) of base window enables the user to select CAN messages (imported from database files) to be edited, tracked or transmitted by diaCAN.These messages can be transmitted once or continuously with the defined period. Similarly, the signals in the imported messages are selected to be observed periodically. On "Log" screen (Figure 4.4), all message traffic on the bus and/or former logs can be displayed as a list. The traffic can be exported as .asc logs. Furthermore up-to-date values of selected messages are tracked continuously.

ECU Operations are handled in "CCP" tab of diaCAN base window. Setting of an ECU is input to diaCAN via .a2l files. When an .a2l file is uploaded, diaCAN saves CAN IDs used by the ECU for received and sent messages. Afterwards, message transmission with ECU is performed according to CAN Calibration Protocol. This transmission process is handled by 3-phased diaCAN-specific CCP control process, as mentioned before, by the "Main Thread".

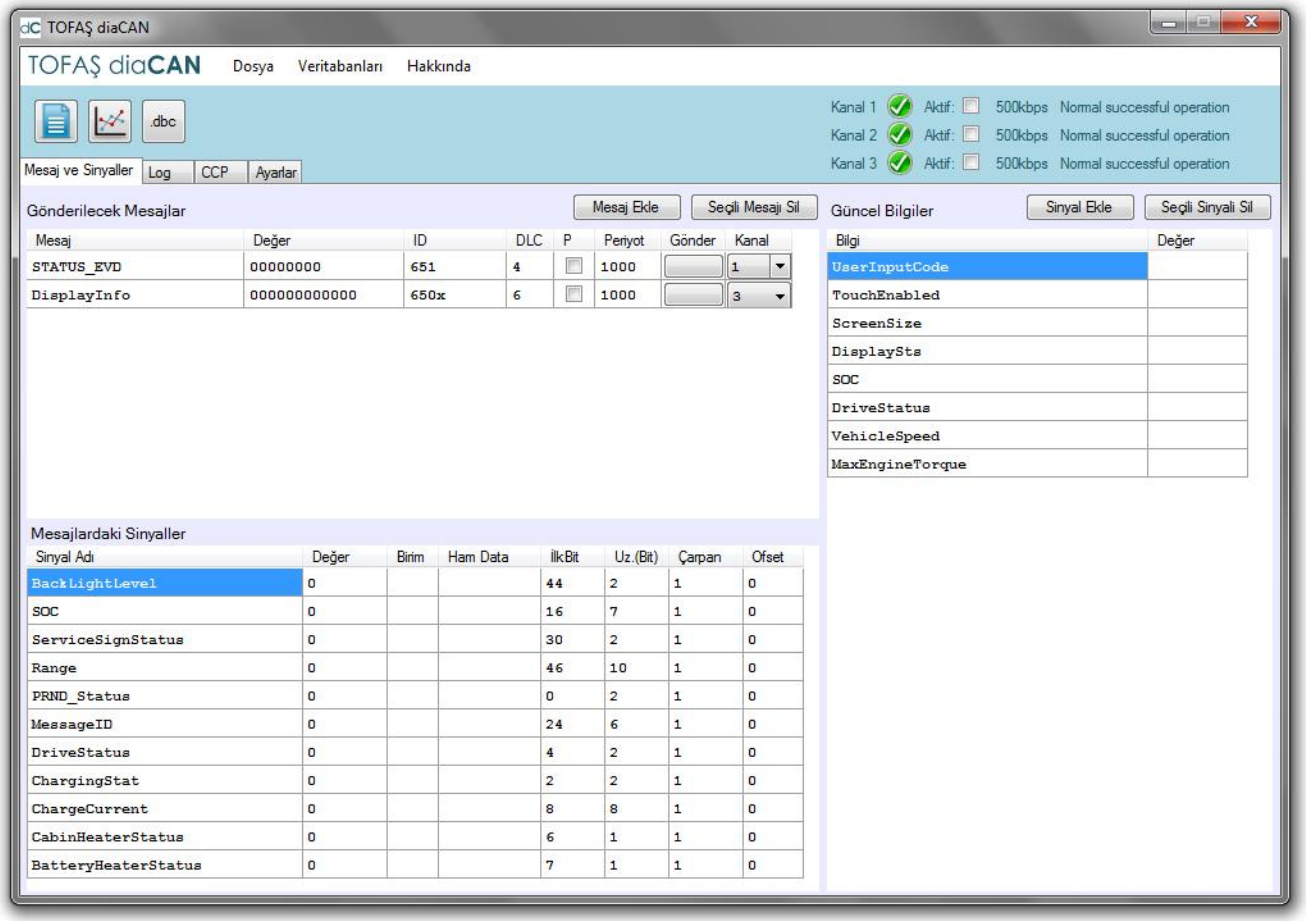
On "Faults" screen of "CCP" tab (Figure 4.5), codes of recent and former faults are visualized on two separate tables. Also, the user can clear former faults saved in ECU.

On "Calibration and Test" screen of "CCP" tab (Figure 4.6), pre-determined configurable settings on an ECU can be calibrated. For example, top speed of the automobile can be set to desired value. For test purposes, test-related components are controlled via track bars. For instance, operation powers of pumps and radiators are changed to observe changes in temperature and pressure values. Consequently cooling performance of the automobile can be simulated in service stations.

Settings of logging process and CAN channels are handled on "Settings" tab (Figure 4.7).

4.2.3 Other Windows

mesajCAN is used to select CAN messages which the user wants to work on. Although all messages defined in the database files are imported to diaCAN, only the selected messages and their last input values are shown. Similarly, *sinyalCAN* is used to choose messages that will be tracked continuously and *grafiCAN*. *bilgiCAN* shows details of the signals tracked. *grafiCAN* plots graphs of selected signals and/or virtual signals generated by their arithmetic products. Info screen includes development and license details of diaCAN for user consideration.



Kanal 1 Aktif: ☐ 500kbps Normal successful operation
Kanal 2 Aktif: ☐ 500kbps Normal successful operation
Kanal 3 Aktif: ☐ 500kbps Normal successful operation

Gönderilecek Mesajlar

Mesaj	Değer	ID	DLC	P	Periyot	Gönder	Kanal
STATUS_EVD	00000000	651	4	☐	1000		1
DisplayInfo	000000000000	650x	6	☐	1000		3

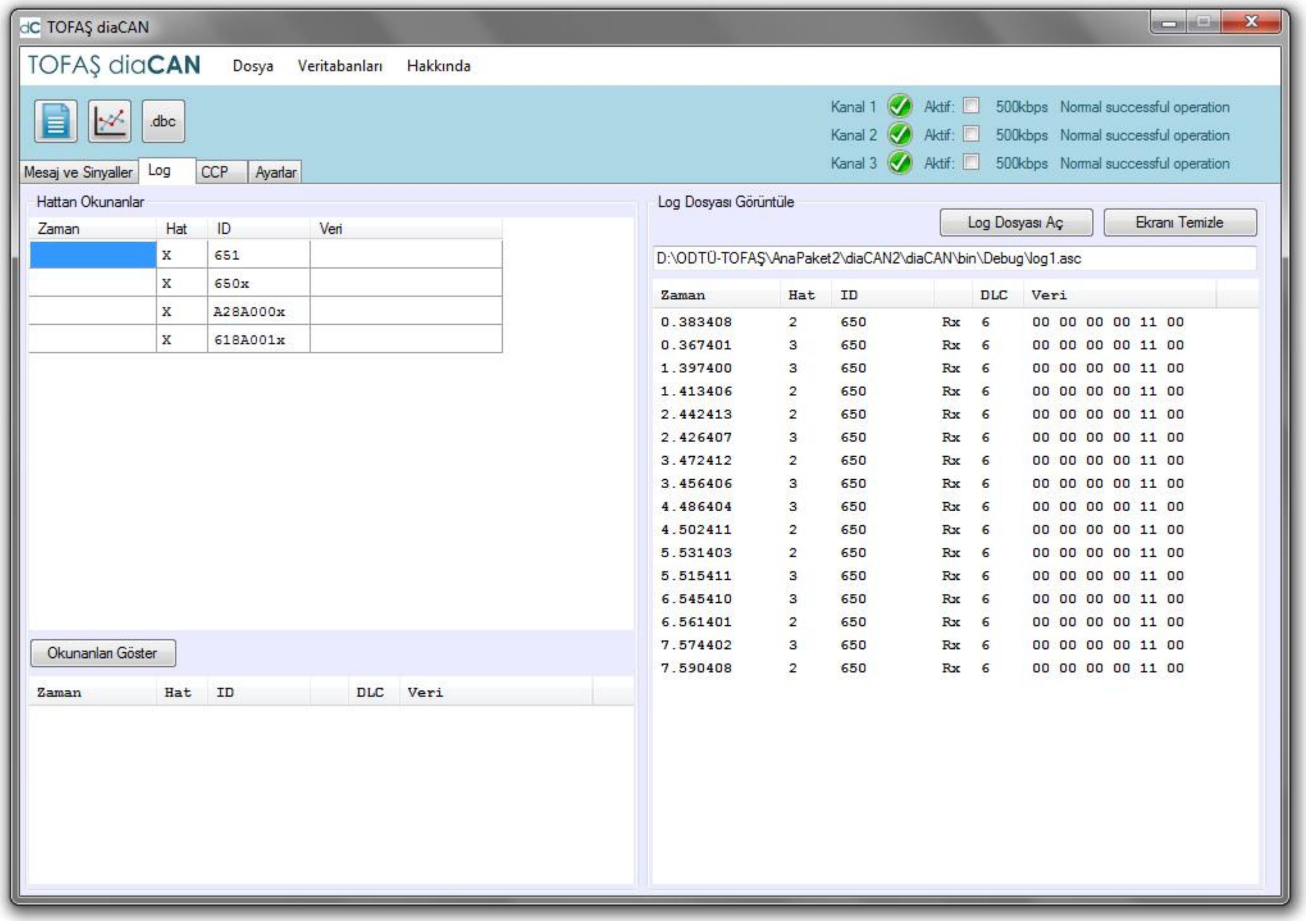
Güncel Bilgiler

Bilgi	Değer
UserInputCode	
TouchEnabled	
ScreenSize	
DisplaySts	
SOC	
DriveStatus	
VehicleSpeed	
MaxEngineTorque	

Mesajlardaki Sinyaller

Sinyal Adı	Değer	Birim	Ham Data	İlkBit	Uz.(Bit)	Çarpan	Ofset
BackLightLevel	0			44	2	1	0
SOC	0			16	7	1	0
ServiceSignStatus	0			30	2	1	0
Range	0			46	10	1	0
PRND_Status	0			0	2	1	0
MessageID	0			24	6	1	0
DriveStatus	0			4	2	1	0
ChargingStat	0			2	2	1	0
ChargeCurrent	0			8	8	1	0
CabinHeaterStatus	0			6	1	1	0
BatteryHeaterStatus	0			7	1	1	0

Figure 4.3: diaCAN base window



Kanal 1 Aktif: ☐ 500kbps Normal successful operation
Kanal 2 Aktif: ☐ 500kbps Normal successful operation
Kanal 3 Aktif: ☐ 500kbps Normal successful operation

Hattan Okunanlar

Zaman	Hat	ID	Veri
	X	651	
	X	650x	
	X	A28A000x	
	X	618A001x	

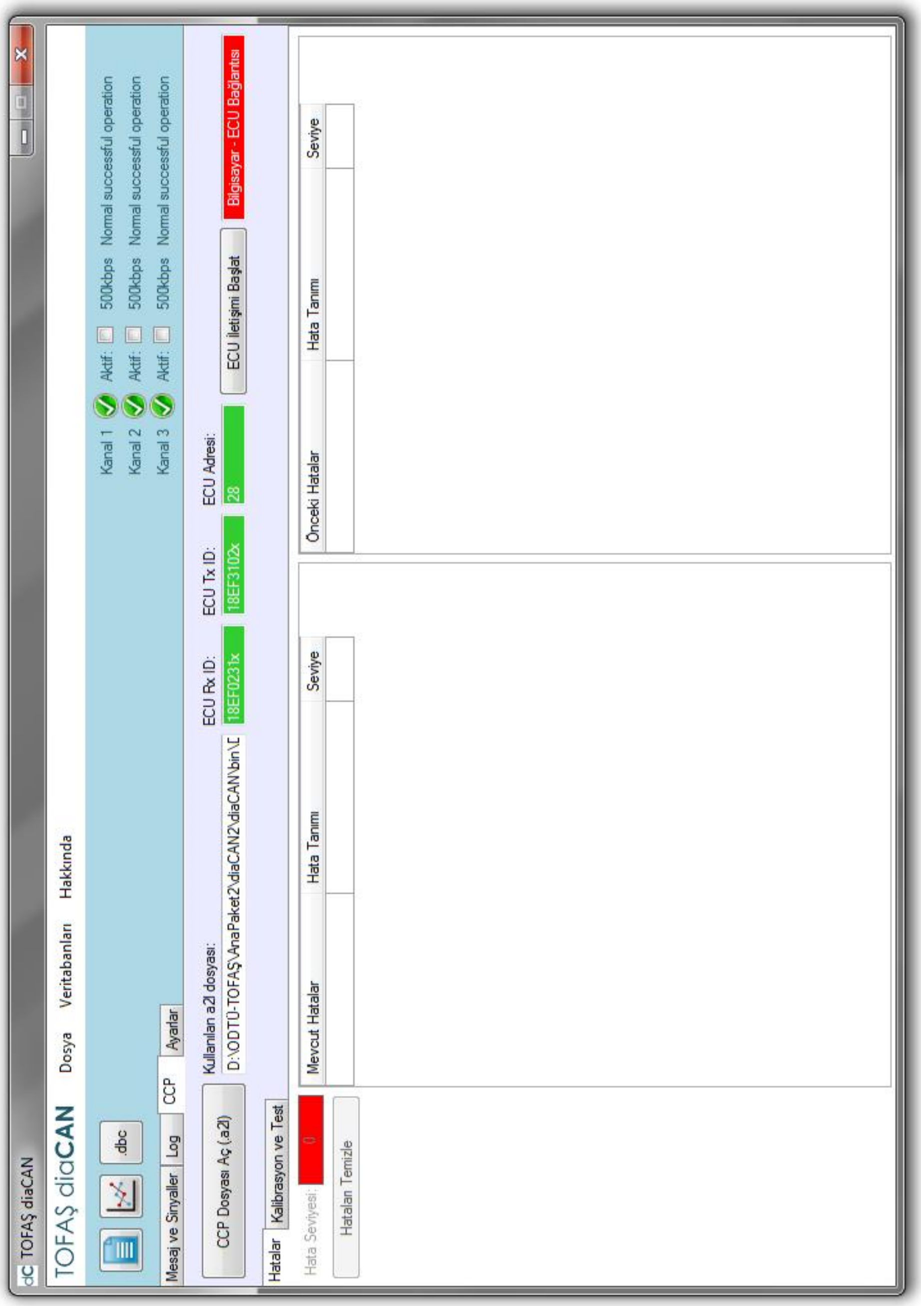
Zaman	Hat	ID	DLC	Veri

Log Dosyası Görüntüle

D:\ODTÜ-TOFAŞ\AnaPaket2\diaCAN2\diaCAN\bin\Debug\log1.asc

```
Zaman      Hat  ID        DLC  Veri
0.383408   2    650   Rx  6    00 00 00 00 11 00
0.367401   3    650   Rx  6    00 00 00 00 11 00
1.397400   3    650   Rx  6    00 00 00 00 11 00
1.413406   2    650   Rx  6    00 00 00 00 11 00
2.442413   2    650   Rx  6    00 00 00 00 11 00
2.426407   3    650   Rx  6    00 00 00 00 11 00
3.472412   2    650   Rx  6    00 00 00 00 11 00
3.456406   3    650   Rx  6    00 00 00 00 11 00
4.486404   3    650   Rx  6    00 00 00 00 11 00
4.502411   2    650   Rx  6    00 00 00 00 11 00
5.531403   2    650   Rx  6    00 00 00 00 11 00
5.515411   3    650   Rx  6    00 00 00 00 11 00
6.545410   3    650   Rx  6    00 00 00 00 11 00
6.561401   2    650   Rx  6    00 00 00 00 11 00
7.574402   3    650   Rx  6    00 00 00 00 11 00
7.590408   2    650   Rx  6    00 00 00 00 11 00
```

Figure 4.4: diaCAN base window



Kanal 1 Aktif: 500kbps Normal successful operation
Kanal 2 Aktif: 500kbps Normal successful operation
Kanal 3 Aktif: 500kbps Normal successful operation

Kullanılan a2l dosyası: D:\ODTÜ-TOFAŞ\AnaPaket2\diaCAN2\diaCAN\bin\L

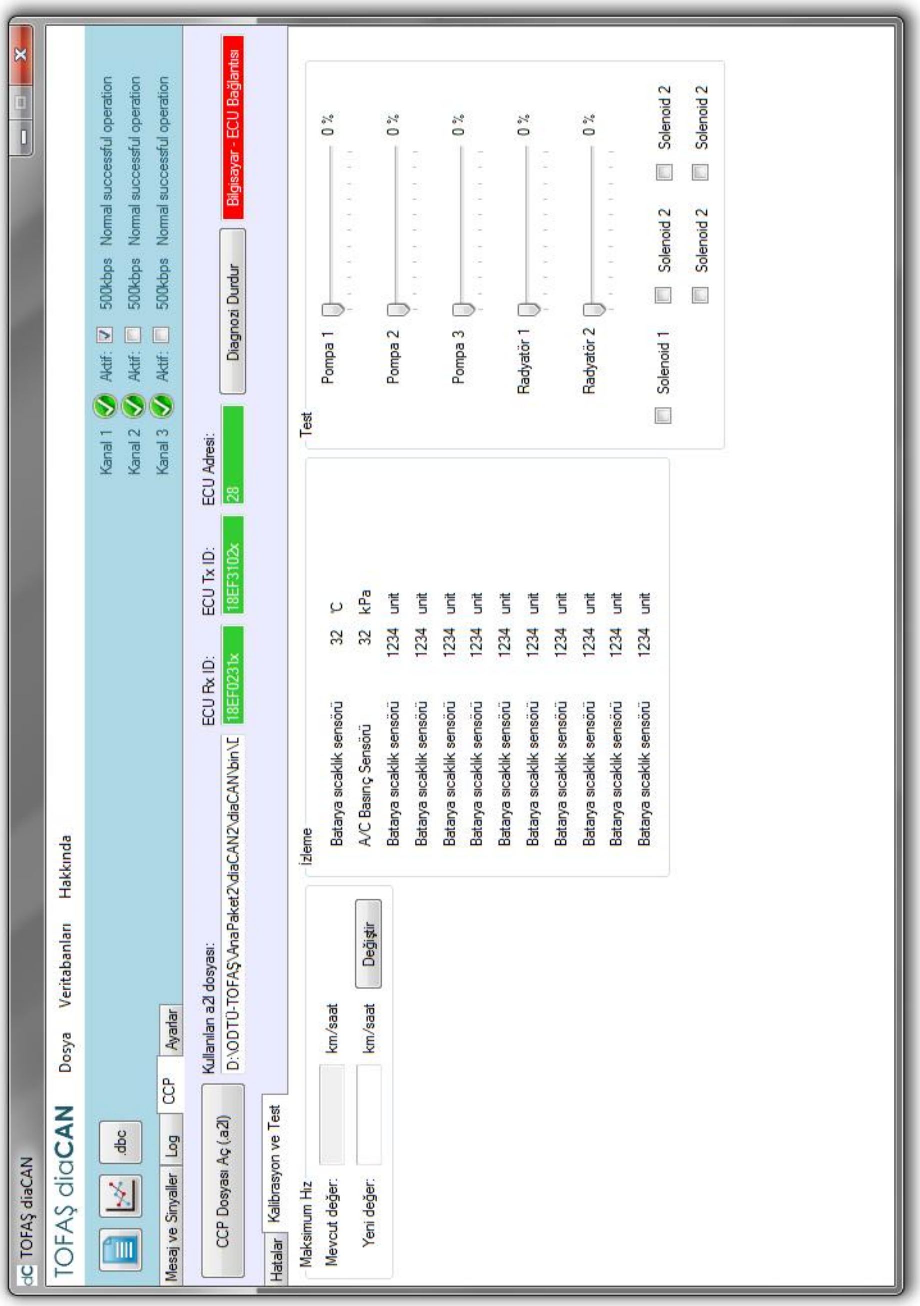
ECU Rx ID: 18EF0231x
ECU Tx ID: 18EF3102x
ECU Adresi: 28

Hata Seviyesi: 0

Mevcut Hatalar	Hata Tanımı	Seviye

Önceki Hatalar	Hata Tanımı	Seviye

Figure 4.5: diaCAN base window



Kanal 1 Aktif: ✓ 500kbps Normal successful operation
Kanal 2 Aktif: ☐ 500kbps Normal successful operation
Kanal 3 Aktif: ☐ 500kbps Normal successful operation

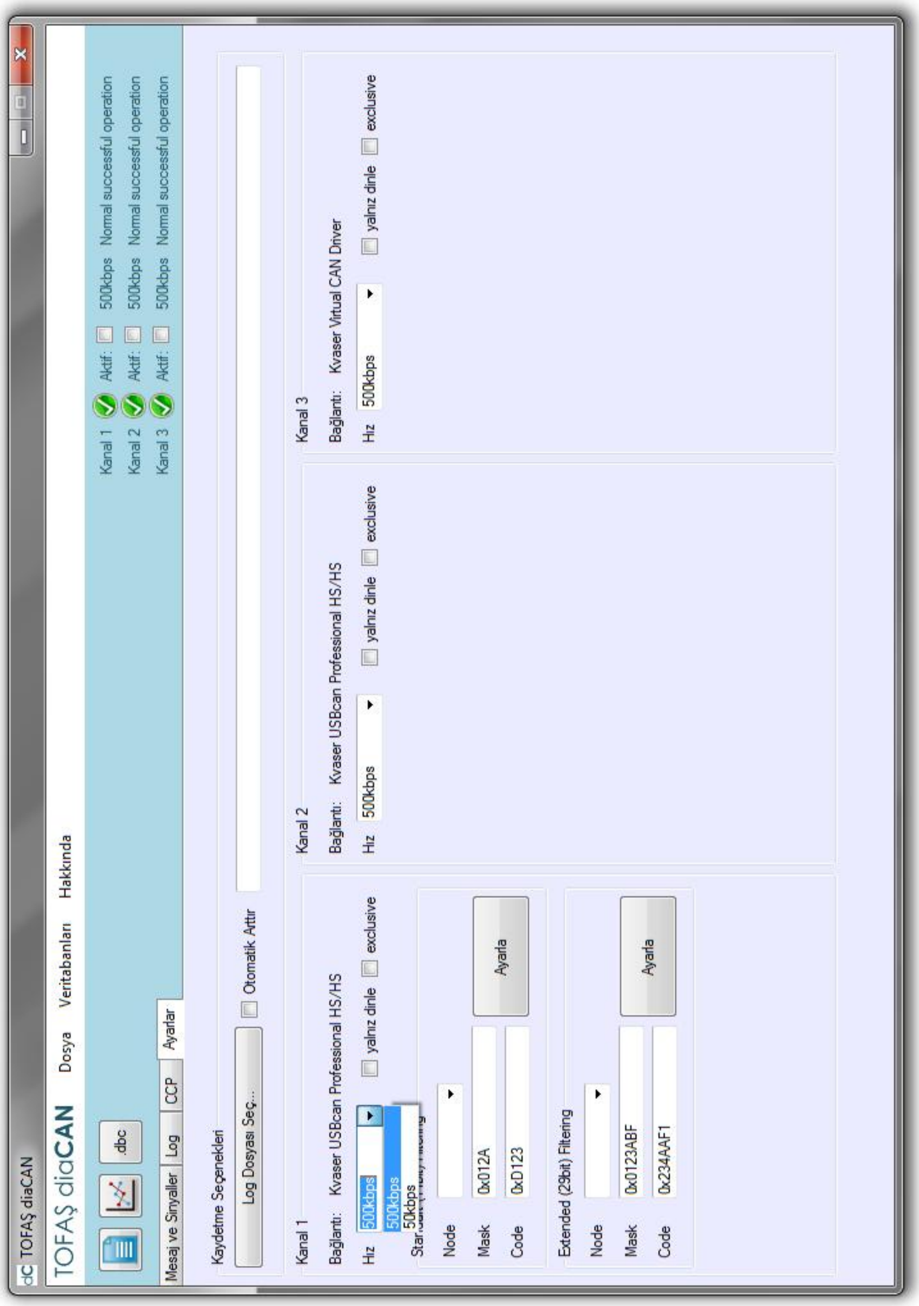
Kullanılan a2l dosyası: D:\ODTÜ-TOFAŞ\AnaPaket2\diaCAN2\diaCAN\bin\
ECU Rx ID: 18EF0231x
ECU Tx ID: 18EF3102x
ECU Adresi: 28
Bilgisayar - ECU Bağlantısı

Maksimum Hız
Mevcut değer: km/saat
Yeni değer: km/saat

Batarya sıcaklık sensörü	32	°C
A/C Basınç Sensörü	32	kPa
Batarya sıcaklık sensörü	1234	unit
Batarya sıcaklık sensörü	1234	unit
Batarya sıcaklık sensörü	1234	unit
Batarya sıcaklık sensörü	1234	unit
Batarya sıcaklık sensörü	1234	unit
Batarya sıcaklık sensörü	1234	unit
Batarya sıcaklık sensörü	1234	unit
Batarya sıcaklık sensörü	1234	unit
Batarya sıcaklık sensörü	1234	unit
Batarya sıcaklık sensörü	1234	unit

Pompa 1 0 %
Pompa 2 0 %
Pompa 3 0 %
Radyatör 1 0 %
Radyatör 2 0 %
☐ Solenoid 1 ☐ Solenoid 2 ☐ Solenoid 2
☐ Solenoid 2 ☐ Solenoid 2

Figure 4.6: diaCAN base window



Kanal 1 Aktif: 500kbps Normal successful operation
Kanal 2 Aktif: 500kbps Normal successful operation
Kanal 3 Aktif: 500kbps Normal successful operation

Kaydetme Seçenekleri
Otomatik Arttır

Kanal 1
Bağlantı: Kvaser USBcan Professional HS/HS
Hız 500kbps yalnız dinle exclusive
500kbps
50kbps
Standard (11bit) Filtering
Node
Mask 0x012A
Code 0xD123
Extended (29bit) Filtering
Node
Mask 0x0123ABF
Code 0x234AAF1

Kanal 2
Bağlantı: Kvaser USBcan Professional HS/HS
Hız 500kbps yalnız dinle exclusive

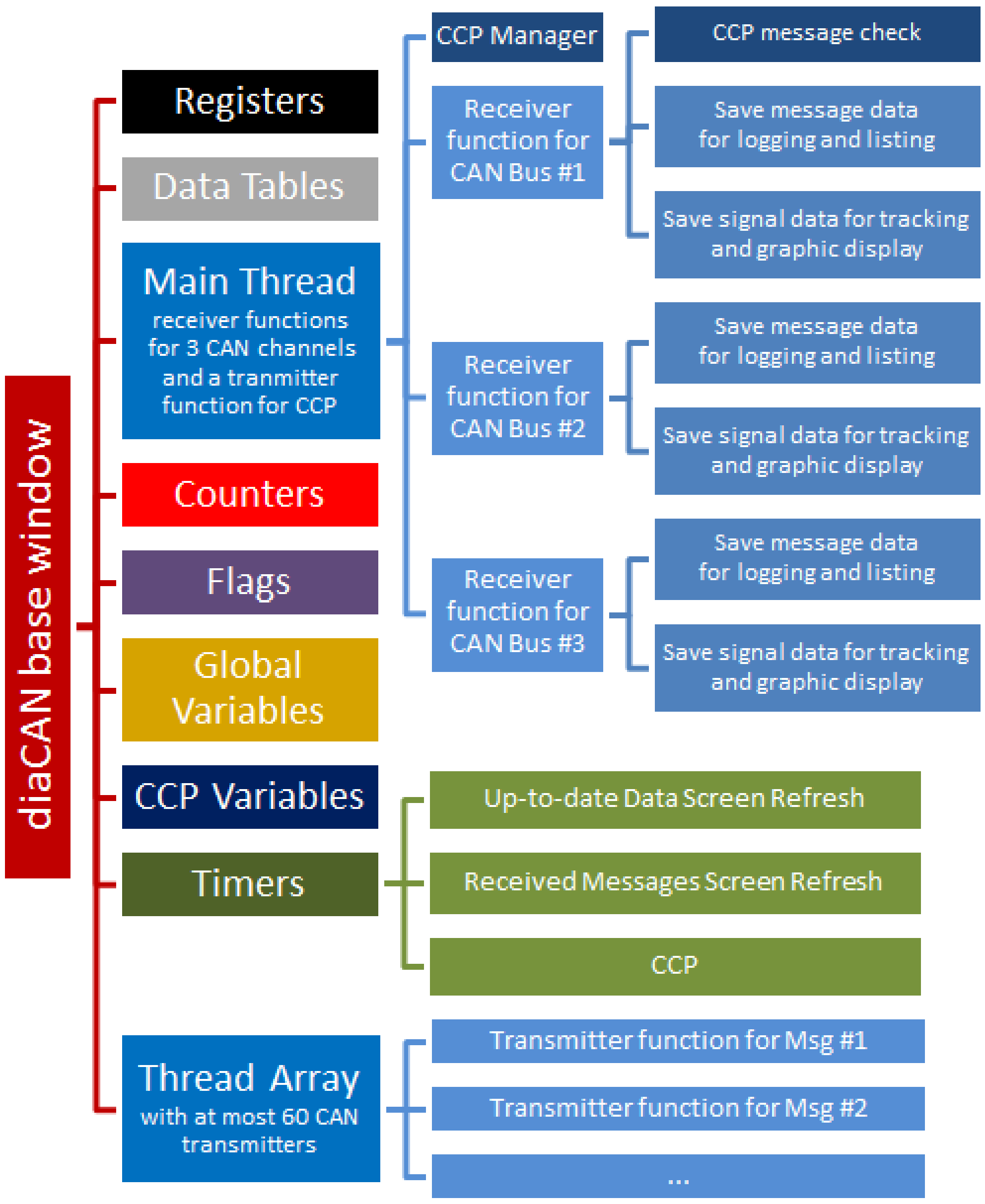
Kanal 3
Bağlantı: Kvaser Virtual CAN Driver
Hız 500kbps yalnız dinle exclusive

Figure 4.7: diaCAN base window

4.3 Detailed Structure

4.3.1 Base Window

Base window of diaCAN (Figure 4.8) can be analyzed in 8 parts: Main thread, thread array, registers, data tables, timers, counters, flags, global variables, and CCP variables.



diaCAN base window
Registers
Data Tables
Main Thread
receiver functions for 3 CAN channels and a tranmitter function for CCP
Counters
Flags
Global Variables
CCP Variables
Timers
Thread Array
with at most 60 CAN transmitters
CCP Manager
CCP message check
Receiver function for CAN Bus #1
Save message data for logging and listing
Save signal data for tracking and graphic display
Receiver function for CAN Bus #2
Save message data for logging and listing
Save signal data for tracking and graphic display
Receiver function for CAN Bus #3
Save message data for logging and listing
Save signal data for tracking and graphic display
Up-to-date Data Screen Refresh
Received Messages Screen Refresh
CCP
Transmitter function for Msg #1
Transmitter function for Msg #2
...

Figure 4.8: diaCAN base window components

4.3.2 Main Thread

One of the critical members of the program is the main receiver thread used to receive CAN messages on the bus and manage CCP communication. For each channel, received messages are saved to a dictionary .NET data structure for listing on the screen andor recording to a log file. Also, signals in received messages are saved to a dictionary for graphical representation. Moreover, if a new value is received for a signal which is added to Up-to-date Data Screen, it is saved to a list for timer-controlled refresh. Pseudo code of the main thread is as follows:

```
Open channels and set channel parameters
Define and initialize variables: (start time, time recording variables, CCP phase register etc.)
— Start of Loop —
// for three channels:
Read channel
If no error is observed and channel is selected as active by the user
    Save data, time, extended CAN-ID info etc. to "All Received Messages" data table
    If CCP is enabled and read message has CAN-ID of CCP Rx (Channel-1 only)
        Update related CCP data on "CCP Messages to Receive" dictionary
    If at least one of the signals of read message is tracked or plotted
        Add new data to "Signal Values vs. Time" and to "Messages to Update Data"
Else
    Get error text and save it to "Channel Info" data table; reopen the channel
// CCP:
If "CCP Messages Queue" is not empty and CCP operations are enabled
    If phase is 1
        CCP counter = Sequence number of CCP messages in the queue
        Get CCP data from "CCP Messages to Transmit" dictionary via CCP Counter
        Transmit CCP message and set phase to 2
    Else if phase is 2
        If "CCP Messages Received" dictionary has an entry for CCP Counter
            Set phase to 3
        Else
            Phase is still 2, wait for reply to last sent CCP message
    Else if phase is 3
        Check "Type of CCP Messages to Receive" entry for CCP message
        Save CCP data to related component (textbox, list etc.) of diaCAN
        Remove CCP entries for the processed CCP message from
            "CCP Messages to Transmit", "Type of CCP Messages to Receive",
            "CCP Messages Received", "CCP Messages Queue"
        Set phase to 1
//For low CPU usage, a 1ms thread sleep can be added to end of each loop.
//If this sleep is added, bus traffic greater than 32,000 Bits/s is no more supported.
— End of Loop —
```

4.3.3 Thread Array

"Thread array" is the 60-thread component of the base window where the number of threads is selected according to the requirements of the vehicle manufacturer. Each member of the array runs the function called "transmit on time". At initialization, the sequence number of the message on "Messages to Transmit" data table is set, "Global Step" counter (see Counters section for details) of diaCAN is get, and a CAN channel handle is created. Then, in each period function operates as follows:

```
If no change is made after last transmission (attributes are the same and no deactivation is made)
    Transmit the message using values of last transmission
Else if transmission is active but a change is made
    Get the number of the transmission channel from "Messages to Transmit"
    Open the CAN channel for created handle
    Get message info from "Messages to Transmit" with internal step number
    Transmit the message
    Close the CAN channel and sleep for the period of the message minus 3ms processing
Else (Message is not checked to be transmitted periodically or channel is inactive or listen-only)
```

```
        Sleep for 50ms //Wait for user entry...
```

Changes in periodic transmission process of messages are tracked via a hidden flag cell in each rows of "Messages to Transmit" data table. When the channel is deactivated or set as listen-only and message attributes are changed (including unchecking periodic transmission checkbox), the flag is set. This flag is cleared as soon as the message is transmitted once by parsing attributes on "Messages to Transmit" data table. Afterwards, only the hidden flag is parsed from this data table in succeeding transmissions. Therefore, CPU usage of the thread is lowered dramatically.

4.3.4 Registers

diaCAN uses a list, a queue and 8 dictionaries as data registers (Figure 4.9).

These registers are used to save

- data obtained from CAN bus,
- mapping of CAN ID's, messages and signals as defined in database files,
- configuration of diaCAN user interface,
- CCP communication data.

"Messages to Display" saves received message's sequence number and CAN related data, in order to enable visualization of all messages received from CAN bus via "Received Messages Screen Refresh" timer. Sequence number is entered to the integer key part of the dictionary where reception time, received channel number, CAN ID, data length code, and data value are recorded to an array of strings. Maximum number of saved messages in this register set to 50,000 (enough to record 10-12 seconds of high traffic). After 50,000 messages, for each arrival of a new message, the oldest message is deleted from this register. Moreover, "Garbage Collector" of .NET is called to release the memory allocated to deleted messages, once in 50,000 message receptions. Therefore RAM usage of diaCAN is kept stable.

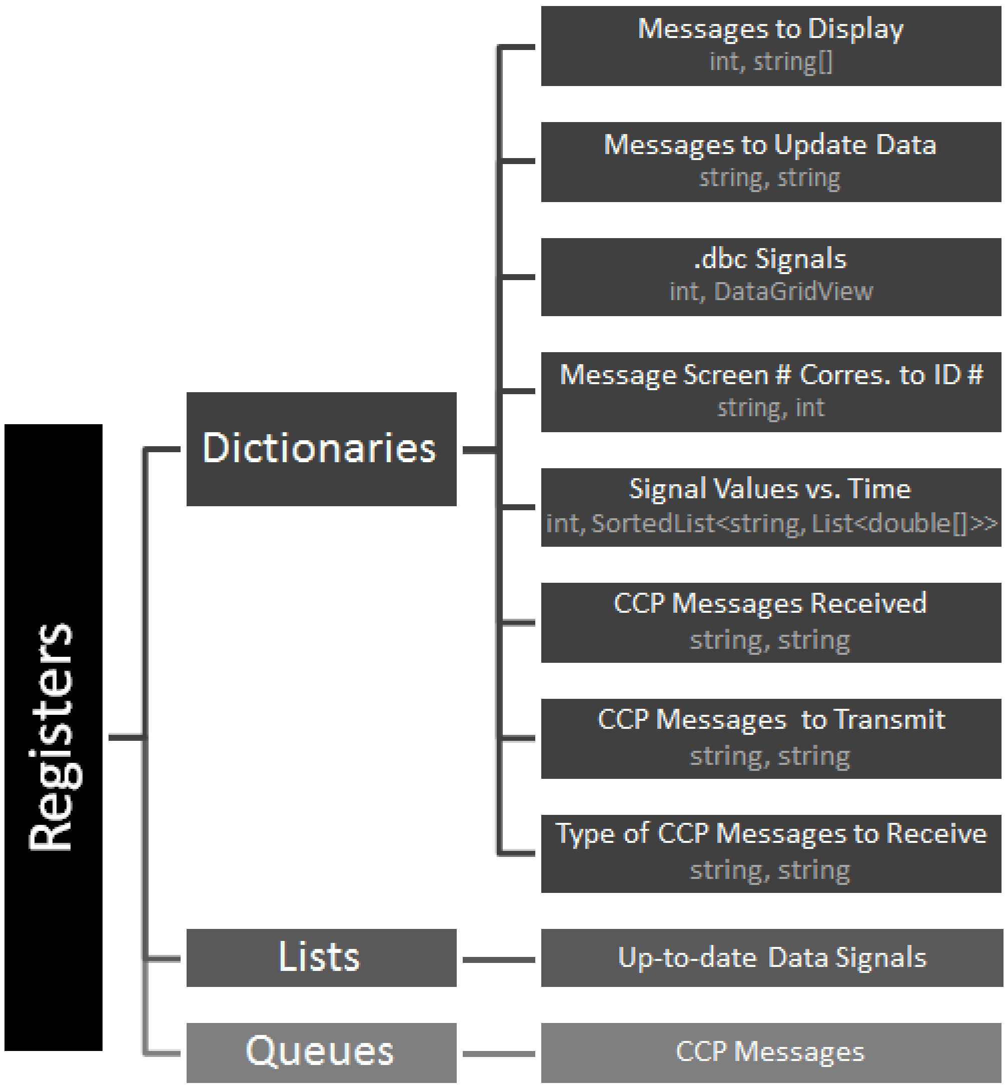


Figure 4.9: Registers

".dbc signals" is a large dictionary keeping the record of all CAN messages and signals imported from database (.dbc) files. "Message Screen #s Corresponding to ID #s" is used for fast matching of selected-modified messages' line numbers on diaCAN window to CAN ID numbers.

"Messages to Update Data" is used to save message data that includes new values of tracked signals. It shortens the time required to update data at each tick of related timer. "Up-to-date Data Signals" list is formed to prevent errors arise from adding signals with same name but with different CAN ID's and properties. When a new signal (data) is want to be added or data tracking or graph plotting, this list is checked.

"CCP Messages" is the queue whose usage is already explained in Main Thread section. It basically controls the two-way handshake transmission of CCP. At every tick of "CCP" timer, calibration, diagnosis and test messages are added to this queue and "CCP Messages to Transmit" dictionary. Related variable (former fault, max. speed etc.) in ECU is saved to "Type of CCP Messages to Receive". Then received reply is saved to "CCP Messages Received" and handled by "Main Thread".

4.3.5 Data Tables

DataGridView and ListView .NET data structures used in diaCAN are grouped under data tables (Figure 4.10).

"Channel Info" is the DataGridView item that displays error condition, baud rate and active status of channels, on the top-right side of diaCAN. Again on the top, "Database Files" shows list of imported .dbc files.

In the first tab of diaCAN, "Messages to Transmit" provide an editable table for CAN messages selected via *mesajCAN*. Name, data value, CAN ID, data length code, periodic transmission checkbox, period, single transmission button and channel to-be-used are columns of the table. "Signals of Messages" is the dynamic editable data table visualizing all signals that are included in the selected message on "Messages to Transmit". Editable properties are name, real value, unit, raw data, first bit in the message, data length in terms of bits and multiplier. Third table in the first tab is "Up-to-date Data". It is used to view the last values of tracked signals.

In the second tab, "Last Values of Updating Messages" monitors last data value of CAN messages used for "Up-to-date Data" refresh. "All Received Messages" shows all messages on the CAN bus from the time it is enabled till it is disabled by the user. "Log File Content" is independent of the bus, since it only displays content of log (.asc and .txt) files.

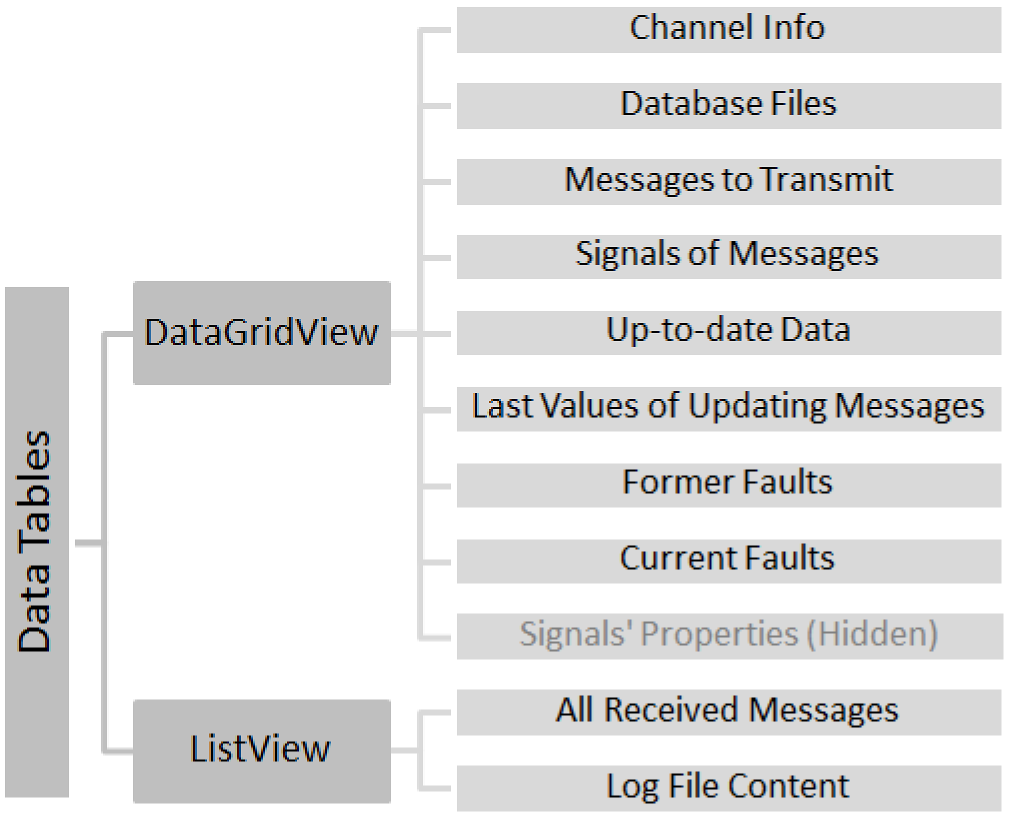


Figure 4.10: Data Tables

"Former Faults" and "Current Faults" reside in the third tab and contain faults determined by ECU and get to diaCAN via CCP transmission.

"Signals' Properties" is a hidden DataGridView element containing properties of signals in "Up-to-date Data". It is used for calculation of real values of tracked data from raw data of messages received.

4.3.6 Timers

diaCAN base window includes three timers with 500 ms intervals. "Up-to-date Data Screen Refresh" is used to load the last received data values to "Up-to-date Data" in first tab and "Last Values of Updating Messages" in second tab of diaCAN. For each signal in hidden "Signals' Properties" table, "Messages to Update Data" dictionary is checked for a new entry. If a new entry exists, full message data is saved to "Last Values of Updating Messages" first. Then signal value is extracted from that value by using hidden "Signals' Properties" table again.

"Received Messages Screen Refresh" is a timer added to diaCAN to lower CPU usage caused by refreshing "All Received Messages" and log recording for every new entry. Received messages are saved to "Messages to Display" dictionary by the main thread as described before. When the timer ticks, messages saved to this dictionary in last time interval are added to "All Received Messages" ListView and log file.

"CCP" timer starts CCP transmission twice a second. For every ECU value demanded, this timer calls a function called `func_ccp_read` (explained in "Base Functions" section) with ECU address of the parameter (obtained from imported .a2l file), data length and data explanation. This function assures the transmission of proper CCP Messages.

4.3.7 Counters

5 global counters are used in diaCAN. Two of them count the total number of messages to display on "All Received Messages" data table, and the sequence number of last message transferred to this data table from "Messages to Display" dictionary. These two counters are used at the tick of "Up-to-date Data Screen Refresh" timer.

Another counter is used for saving successive log files with ascending file numbers (logfile01.asc, logfile02.asc, logfile03.asc...). At the end of each record, this counter is incremented for the filename of next log.

The function of CCP Counter is explained in Main Thread and Timers sections.

The last counter is "Global Step". When the database is cleared by the user, the CAN messages imported from .dbc files are also inactivated. However, message transmitter threads may not be inactivated instantly due their large periods. In order to enable transmission of messages to-be-imported, a new thread is created while former thread is being aborted (Figure 4.11). This reloading process is counted by "Global Step". When a former thread calls values on "Messages to Transmit" data table, no value is returned due to inconsistency in step numbers.

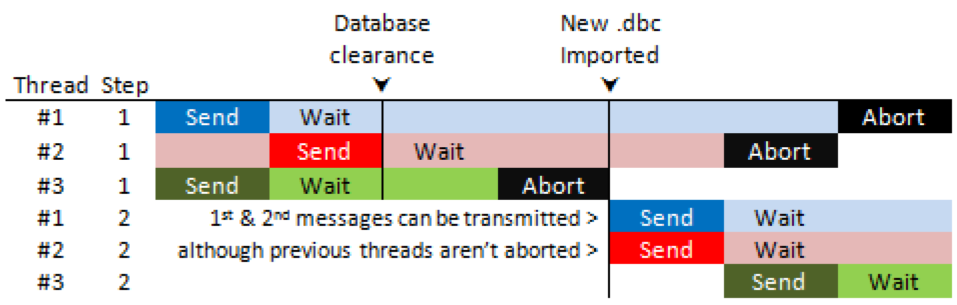


Figure 4.11: Use of Global Step

4.3.8 Flags

diaCAN uses two kinds of flags: Channel flags and program flags(Figure 4.12). For each channel, a channel flag indicates whether channel is listened-only or not (message transmission is enabled).

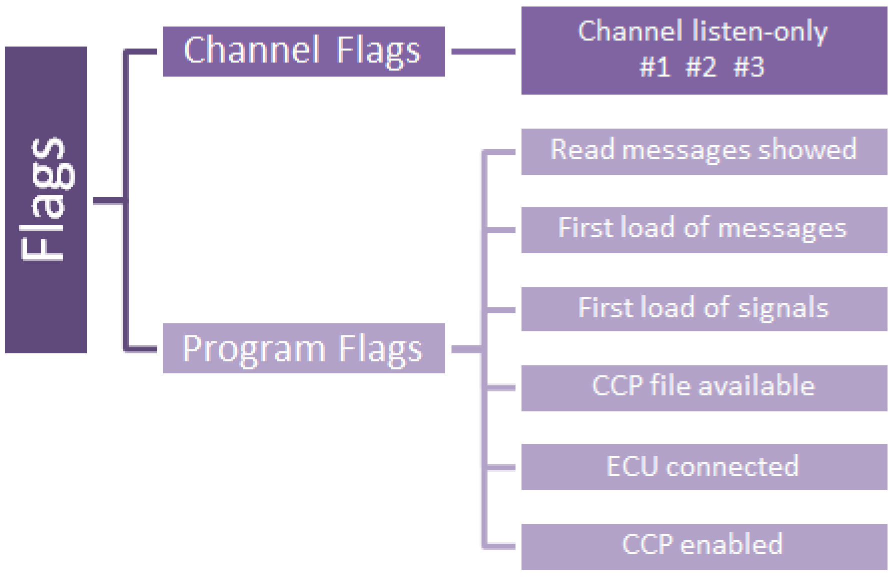


Figure 4.12: Flags

"Read messages showed" denotes whether the user has wanted to see all messages received from active channels. "Received Messages Screen Refresh" timer checks this flag to transfer CAN messages recorded to "Messages to Display" dictionary.

"First load of messages" and "first load of signals" are used to differentiate two types of CAN message configurations: Imported from database files and entered by user. If these flags are not set, user interface controls are enabled when a value change is observed in data tables.

Other flags are useful in establishing CCP communication between diaCAN and ECU. When user imports a valid .a2l file, "CCP file available" flag is set. Afterwards, user can start the communication by pressing related button in "CCP" tab of diaCAN. If ECU replies to this request properly, "ECU connected" flag is set and user is authorized to enable CCP operations defined in diaCAN. When these operations are enabled, "CCP enabled" flag is set to inform "CCP" timer, which handles the communication.

4.3.9 Global Variables

Although many variables are created within functions throughout the run of diaCAN; some variables(Figure 4.13) are defined globally to control more than one function.

4.3.9.1 Message Variables

Message variables are used in changes of message & signal values, and log process.

When the user clicks a CAN message on "Messages to Transmit" data table, "Line # of selected message" and "Data value of selected message" variables changed accordingly. First variable is used in modifying signal values of changed message, while the second variable is used to reset data value when user enters an improper value. It is also used to delete a message from "Messages to Transmit" data table.

"Line # of selected signal" saves the selection of user on "Up-to-date Data" data table. When the user clicks "Delete selected signals" button on the top of this data table, "Line # of selected signal" variable is used for required operations.

For log file generation, "Name of log file", "Start time of diaCAN", and "Start time of logging" variables are used. First variable is used to keep address and name of current log file. Latter two variables are used to calculate time offset in logging activation, so that log file is recorded as if the logging is started at time 0.

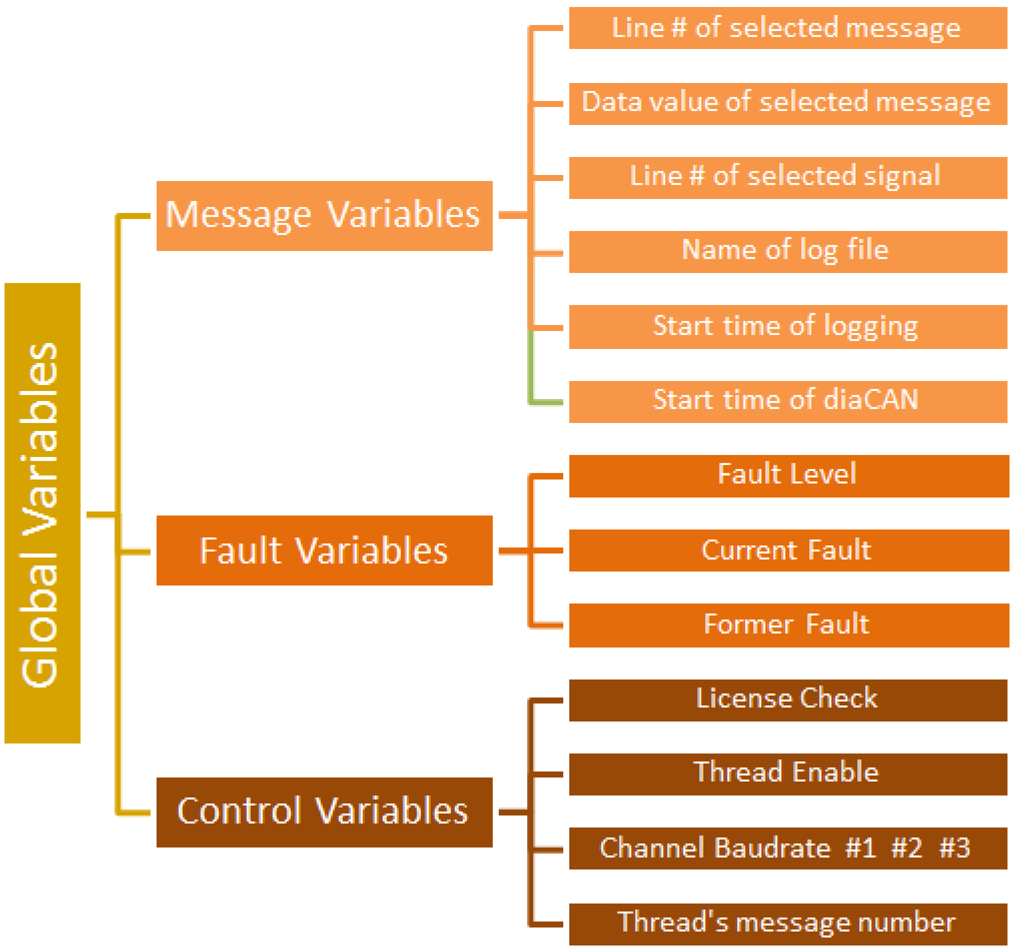


Figure 4.13: Global Variables

4.3.9.2 Control Variables

"License Check" is the variable whose value is transferred from *acilirCAN*. It indicates whether a licensed hardware is connected or not.

"Channel Baudrate" is an array of length 3, keeping automatically sensed or user-selected baud rates of channels. In transmission and reception of CAN messages, these variables are used in setting channel parameters.

"Thread Enable" is checked by main thread perpetually since it used to disable/enable it. When the baud rate of a channel is wanted to be changed by the user, "Thread Enable" is set false and main thread stops its operation. After proper modification of parameters, this variable is set true again and main thread continues to run.

"Thread's message number" is used to pass CAN messages' line numbers in "Messages to Transmit" to corresponding threads in "Thread Array". This transfer is performed when CAN messages are imported to diaCAN via .dbc files and new threads are to be initialized for them.

4.3.9.3 Fault Variables

diaCAN demands three types of fault data from ECU. ECU determines the level of current faults by using the fault with highest importance. This data is get and saved to "Fault Level". ECU also transmits current and former faults one by one. These faults are recorded to "Current Level" and "Former Level" variables temporarily and transferred to related data tables by the main thread.

4.3.10 CCP Variables

For every parameter to be read via CCP, an uninitialized global variable is defined to keep ECU address of the parameter. These variables are set when an ECU configuration (.a2l) file is imported. Items in "CCP" tab of diaCAN and "CCP" timer use these variables to carry out calibration, test and diagnosis operations.

4.3.11 Base Window Items' Functions

In addition to above classified components, Base window has a variety of buttons, comboboxes, textboxes, checkboxes, menu strips, open file dialogs and save file dialogs. Crucial items of these types will be explained in this section.

4.3.11.1 Top Items

At the top of diaCAN base window (Figure **??**), 3 menu strips, 2 square checkbox buttons, 1 button and a tab control exist.

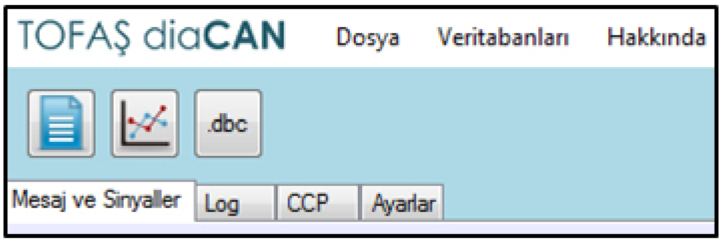


Figure 4.14: Items at the top of diaCAN base window

Leftmost square button (with visual of a document) is a checkbox. It starts and stops record of received CAN messages to log files. After the button is checked, "Start time of diaCAN" variable is used to calculate relative "Start time of logging" variable and "Received Messages Screen Refresh" timer's ticking function starts to record all received messages to the log file.

The square box in the middle opens diaCAN window, where the rightmost square box opens "Open .dbc File Dialog". When a proper .dbc file is selected, this dialog imports message and signal information to related data tables, registers, *mesajCAN* and *sinyalCAN* via a function called "Read Stream" (explained in "Base Functions" section). Moreover, message transmitter threads in "Thread Array" are initialized by this dialog.

Menu strips at the top of diaCAN are given in (Figure 4.15).

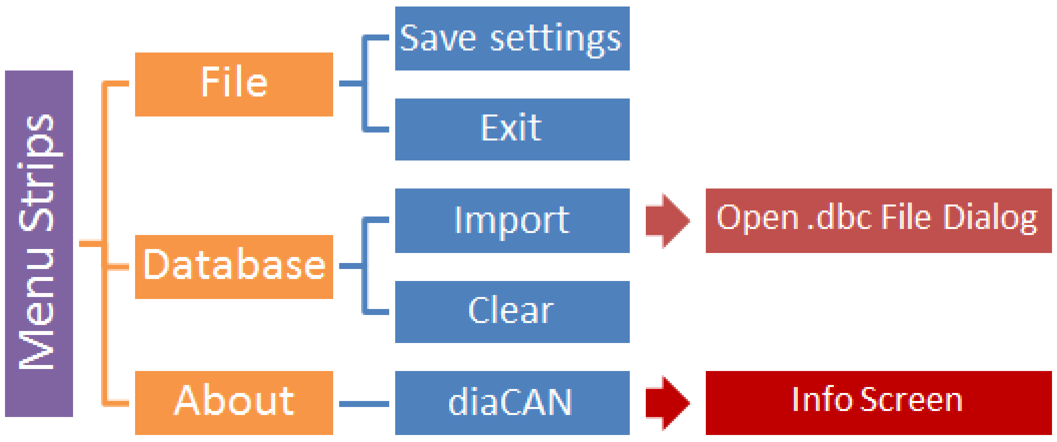


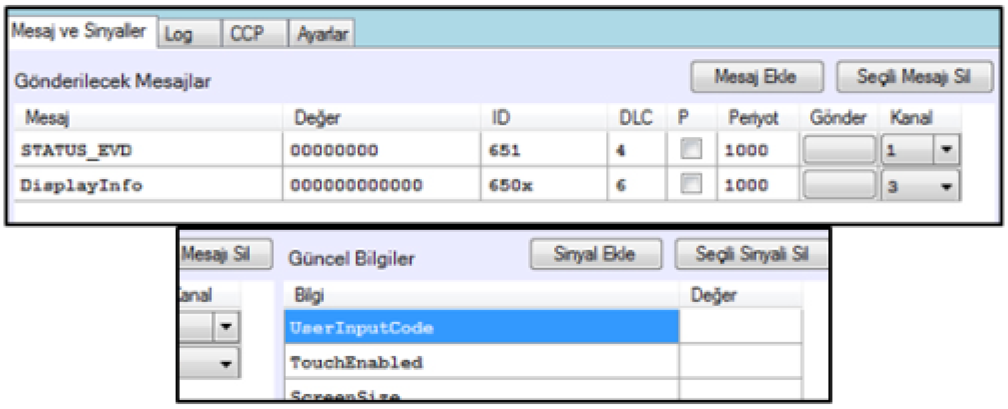
Figure 4.15: Menu Strips

"Save settings" entry saves paths of imported database files, screen numbers of displayed messages and tracked signals' names, message and signal numbers to a text file. This file has a default name and checked in every start-up of diaCAN to use loaded import and user interface settings.

"Exit" resets "Thread Enable" flag and aborts all threads in diaCAN. Afterwards diaCAN is closed. "Database import" opens "Open .dbc File Dialog". "Clear database" option aborts threads in "Thread Array", increases "Global Step" counter and clears all data tables, registers, *mesajCAN* and *sinyalCAN* entries. "About > diaCAN" opens Info Screen.

4.3.11.2 Messages and Signals Tab

In this tab (Figure 4.16), 4 buttons exist. "Add Message" button opens *mesajCAN* where "Delete Selected Message" removes selected message from "Messages to Transmit" data table. Similarly, "Add Signal" button opens *sinyalCAN* where "Delete Selected Signal" removes selected signal from "Up-to-date Data" data table.



Gönderilecek Mesajlar

Mesaj	Değer	ID	DLC	P	Periyot	Gönder	Kanal
STATUS_EVD	00000000	651	4	☐	1000		1
DisplayInfo	000000000000	650x	6	☐	1000		3

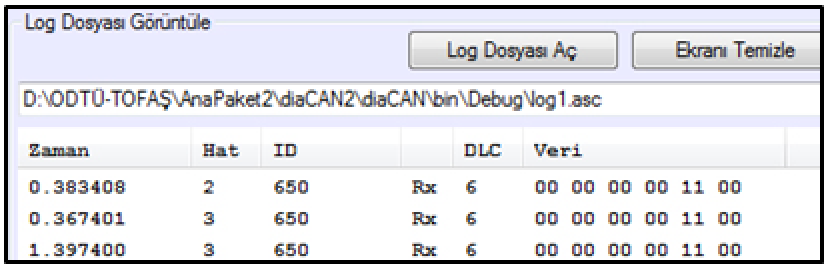
Güncel Bilgiler

Bilgi	Değer
UserInputCode	
TouchEnabled	
ScreenSize	

Figure 4.16: Items in the first tab of diaCAN base window

4.3.11.3 Log Tab

This tab (Figure 4.17) includes two buttons: "Open Log File" and "Clear Screen". First button opens "Open Log File Dialog" which transfers records in selected .asc file to "Log File Content" data table. On the other hand, second button deletes all entries in this data table.



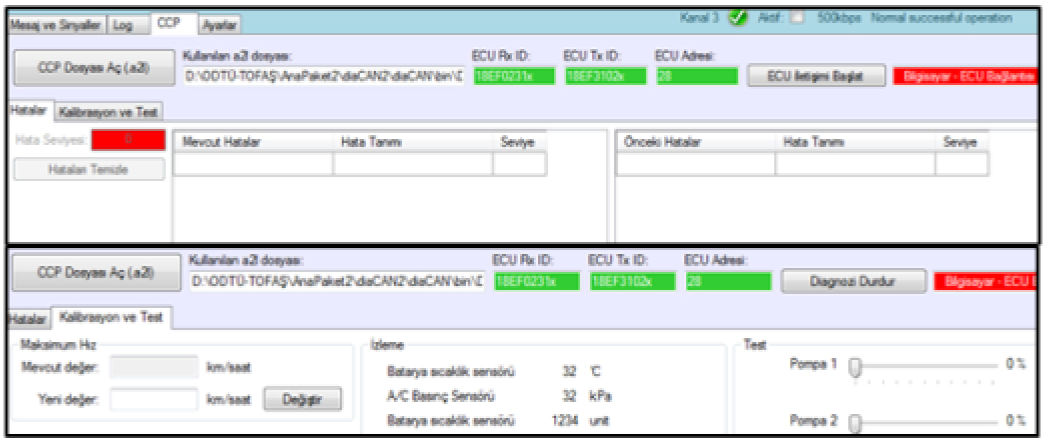
D:\ODTÜ-TOFAŞ\AnaPaket2\diaCAN2\diaCAN\bin\Debug\log1.asc

```
Zaman        Hat   ID          DLC   Veri
0.383408     2     650     Rx  6     00 00 00 00 11 00
0.367401     3     650     Rx  6     00 00 00 00 11 00
1.397400     3     650     Rx  6     00 00 00 00 11 00
```

Figure 4.17: Items in the second tab of diaCAN base window

4.3.11.4 CCP Tab

This tab (Figure 4.18) has a variety of items. Items at the top of tab are used to control CCP communication. "Open CCP File" button opens "Open CCP File Dialog" which imports ECU configuration via an .a2l file. This dialog transfers ECU addresses of parameters to be read, and set CCP variables accordingly. It also writes addresses of ECU, ECU Transmitter, and ECU Receiver to corresponding textboxes in CCP Tab. These textboxes are used to get address information, throughout CCP communication.



D:\ODTÜ-TOFAŞ\AnaPaket2\diaCAN2\diaCAN\bin\

ECU Rx ID: 18EF0231x | ECU Tx ID: 18EF3102x | ECU Adresi: 28

Hata Seviyesi: 0

Batarya sıcaklık sensörü 32 °C
A/C Basınç Sensörü 32 kPa
Batarya sıcaklık sensörü 1234 unit

Pompa 1 0 %
Pompa 2 0 %

Figure 4.18: Items in the third tab of diaCAN base window

Name, definition and level of all possible faults are automatically imported to "Current Faults" and "Former Faults" data tables from "dtclist.xls" file which should be in the same directory with diaCAN executable file.

"Start ECU Connection" button starts the communication between ECU and computer by transmitting proper CCP messages. Then, button text is changed as "Stop ECU Connection". And it is used to terminate the connection:

```
If "Start ECU Connection" is pressed
    Change text to "Stop ECU Connection"
    Form data string "s":
        01 ("Connect" command code)
        + CCP Counter (byte)
        + Station address
        + 00000000 (don't care)
    Add CCP Counter (byte) & data string pair to "CCP Messages Transmitted" dictionary
    Add CCP Counter (byte) & "0" pair to "Type of CCP Messages to Receive" dictionary
    Add CCP Counter (byte) to "CCP Messages" queue
    Set "CCP Enabled" flag
    Increase CCP counter and get & save it as a byte string again
    Start "CCP" timer
Else
    Change text to "Start ECU Connection"
    Clear CCP data tables:  "CCP Messages "CCP Messages to Transmit" "Type of CCP
Messages to Receive" "CCP Messages Received"
    Form data string "s":
        07 ("Disconnect" command code)
        + CCP Counter (byte)
        + 00 (Temporary disconnection)
        + 00 (don't care)
        + Station address
        + 000000 (don't care)
    Send a CAN message with CAN ID of ECU receiver, and data value of string s
    Reset "CCP Enabled" flag
    Set CCP counter to 1
    Stop "CCP" timer
```

Textbox (with text "Computer - ECU Connection") at the right of "Start ECU Connection" button turns to green when the data connection is established and to red when the connection is over.

In "Faults" sub-tab, "Clear Faults" button exists. This button calls `func_ccp_set` function (explained in "Base Functions" section) 4 times: For set "Clear All Faults New Value", set "Clear All Faults Override Enable", reset "Clear All Faults Override Enable" and reset "Clear All Faults New Value" variables of ECU one by one:

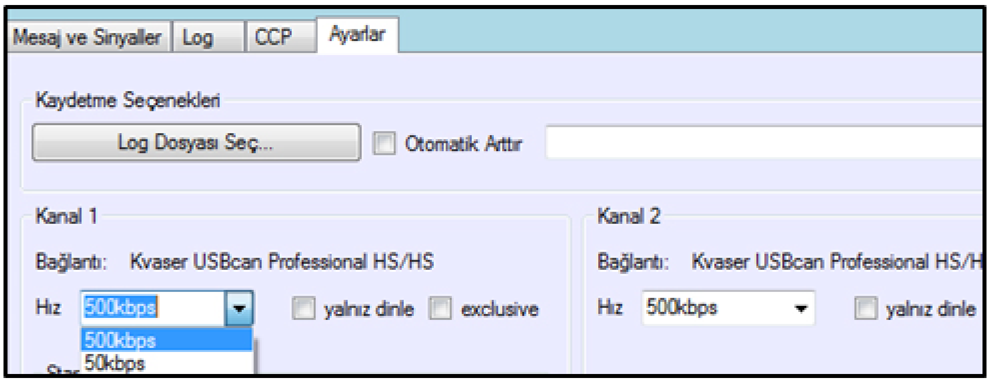
- func_ccp_set(Clr_new, "01", "01");
- func_ccp_set(Clr_ovr, "01", "01");
- func_ccp_set(Clr_ovr, "01", "00");
- func_ccp_set(Clr_new, "01", "00");

"Calibration and Test" sub-tab is divided to 3 parts: Calibration, Tracking, and Test. Buttons, textboxes, and trackbars in this sub-tab operate by using `func_ccp_set` and `func_ccp_read` and functions and override, change or get values on ECU.

4.3.11.5 Settings Tab

Items in this tab(Figure 4.19) are used to control CAN channel and logging parameters. "Choose Log File" button is used to select the path and name of the log file. If the check-box of "Automatically Increase" is checked, successive log files are saved with ascending file numbers.

"Speed" comboboxes are selected to change baud rate of channels. These items change value of "Channel Baudrate" control variables. Similarly, "Listen Only" checkboxes modify "Channel Listen-Only" flags. "Exclusive" checkboxes and message filtering items are currently unavailable.



Bağlantı: Kvaser USBcan Professional HS/HS

Hız 500kbps — yalnız dinle, exclusive

500kbps
50kbps

Bağlantı: Kvaser USBcan Professional HS/H

Hız 500kbps — yalnız dinle

Figure 4.19: Items in the fourth tab of diaCAN base window

4.3.12 Base Functions

"Read Stream" is a function used by "Open .dbc File Dialog", to import message and signal information from .dbc files.

```
Loop until the document has no more lines
    Read a line
    If "BO_ " exists in line, i.e. details of a message are given
        Split the line by whitespaces and transfer to a string array
        Add a new hidden line to "Messages to Transmit" by using this string array
            > Data value is message size times "00"
            > Period value is set to "1000 ms" by default
            > Transmission channel is set to "1" by
        Add the name of the message to sinyalCAN and mesajCAN
        Add new entry to ".dbc Signals"
            > Pair of line # of selected message & new empty DataGridView
        Add new entry to "Message Screen # Corres. to ID #"
        > Pair of line # of selected message & CAN ID
    If "SG_ " exists in line, i.e. details of a signal are given
        Split the line by whitespaces and transfer to a string array
        Add signal information array to corresponding DataGridView in ".dbc Signals"
        Add the name of the signal to sinyalCAN
```

`func_ccp_read` is used to read values on ECU via CCP:

```
Get & save CCP counter as a byte string
Form data string "s":
    02 ("Set Memory Transfer Address" command code)
    + CCP Counter (byte)
    + 00 (MTA number)
    + 02 (Address extension)
    + Parameter address
Add CCP Counter (byte) & data string pair to "CCP Messages Transmitted" dictionary
Add CCP Counter (byte) & "0" pair to "Type of CCP Messages to Receive" dictionary
Add CCP Counter (byte) to "CCP Messages" queue
Increase CCP counter and get & save it as a byte string again
Form data string "s":
    04 ("Data Upload" command code)
    + CCP Counter (byte)
    + Data length
    + 0000000000 (don't care)
Add CCP Counter (byte) & data string pair to "CCP Messages Transmitted" dictionary
Add CCP Counter (byte) & user defined data explanation pair to "Type of CCP Messages to
Receive" dictionary
Add CCP Counter (byte) to "CCP Messages" queue
```

`func_ccp_set` is used to set values on ECU via CCP:

```
Get & save CCP counter as a byte string
Form data string "s":
    02 ("Set Memory Transfer Address" command code)
    + CCP Counter (byte)
    + 00 (MTA number)
    + 02 (Address extension)
    + Parameter address
Add CCP Counter (byte) & data string pair to "CCP Messages Transmitted" dictionary
Add CCP Counter (byte) & "0" pair to "Type of CCP Messages to Receive" dictionary
Add CCP Counter (byte) to "CCP Messages" queue
Increase CCP counter and get & save it as a byte string again
Form data string "s":
    03 ("Data Download" command code)
    + CCP Counter (byte)
    + Data length
    + Data to be transferred (up to 5 bytes)
Add CCP Counter (byte) & data string pair to "CCP Messages Transmitted" dictionary
Add CCP Counter (byte) & user defined data explanation pair to "Type of CCP Messages to
Receive" dictionary
Add CCP Counter (byte) to "CCP Messages" queue
```

4.3.13 grafiCAN

grafiCAN window(Figure 4.20) is dedicated to graphical screening of changes in signal values. The user can choose signals predefined for a CAN message or generated via arithmetic operations on other signals. Arithmetic signal computation enables visualization of values which are not directly measured. For instance, the power generated by a sensor can be obtained via multiplication of its voltage and current values. *grafiCAN* can display many graphics at the same time, on the same plane. That eases the comparison and correlating among signals.

grafiCAN has 6 important functions/items: "Sketching Selection" data table, "Arithmetic Signals" data table, "Signal Adder" function, "Graph" Timer, "Evaluator" function, and "Signal Chart" sketching area.

"Sketching Selection" data table takes place on the top left corner of *grafiCAN* window. It saves properties of every signal on "Up-to-date Data" data table of diaCAN base window and arithmetic signals created by user. These properties are signal's name, message's line #, signal's sequence # and a checkbox for sketching this signal. Signal name and checkbox are displayed on screen for user selection while other variables are hidden to be used by functions.

"Arithmetic Signals" data table keeps evaluation parameters of user-created virtual signals. Columns of this data table are as follows:

Message's CAN ID - Message's Line # - Signal's Name - Signal's Unit - Coefficient 1 - Signal 1 - Operation 1 - Coefficient 2 - Signal 2 - Operation 2 - Coefficient 3 - Operation 3 - Coefficient 4

where the overall operation has the form

[Coef 1 (Signal 1)] oper 1 [Coef 2 (Signal 2)] oper 2 [Coef 3] oper 3 [Coef 4]

"Signal Adder" is a function used to record arithmetic signals to "Arithmetic Signals" data table. In addition to this record, it adds the processed signal to "Sketching Selection" data table. Message's line number is set to "99" where it can be 60 at most for a real message. Signal's sequence number is set to its sequence number on "Arithmetic Signals" data table. Moreover, a graphic series with the name of user-created virtual signal is added to "Signal Chart".

"Graph" timer ticks once a second to refresh "Signal Chart". If the checkbox of a real signal is checked, the latest data pair of "Signal Values vs. Time" dictionary of diaCAN base window is added to sketch of latest graph of the signal. As a result of this, in each tick, not all values but only the latest values of signals are added to sketch. If the checked signal is an arithmetic signal, variables in "Arithmetic Signals" data table, and "Signal Values vs. Time" dictionary records of operant signals are parsed as an expression to "Evaluator" function. The output of this function is used to update arithmetic signal's graph. However, no update is performed if the difference between reception time of operant signals is bigger than 1 second.

"Evaluator" function works as follows:

An empty data table and a data column with parsed expression is created:

```
var loDataTable = new DataTable();
var loDataColumn = new DataColumn("Eval", typeof(double), expression);
```

This data column is added to data table:

```
loDataTable.Columns.Add(loDataColumn);
loDataTable.Rows.Add(0);
```

Value-time pair is returned as the output of function:

```
return (double)(loDataTable.Rows[0]["Eval"]);
```

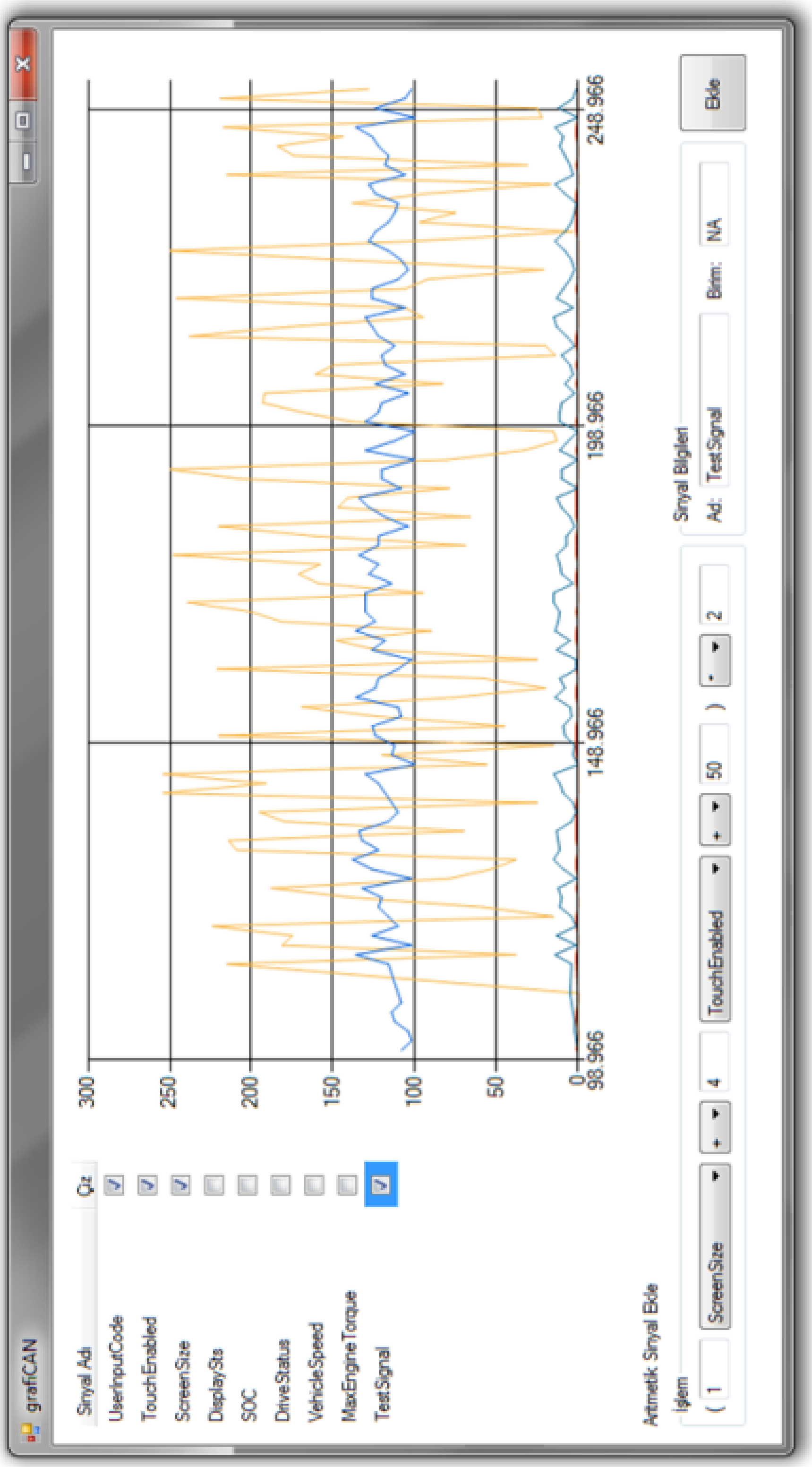


Figure 4.20: *grafiCAN* window

4.3.14 mesajCAN

mesajCAN is used to select messages that will be edited and transmitted. When a database file is imported, all messages are transferred to diaCAN. These messages are saved hidden to "Messages to Transmit" data table. The user can change appearance option of these messages by using *mesajCAN* interface(Figure 4.21).

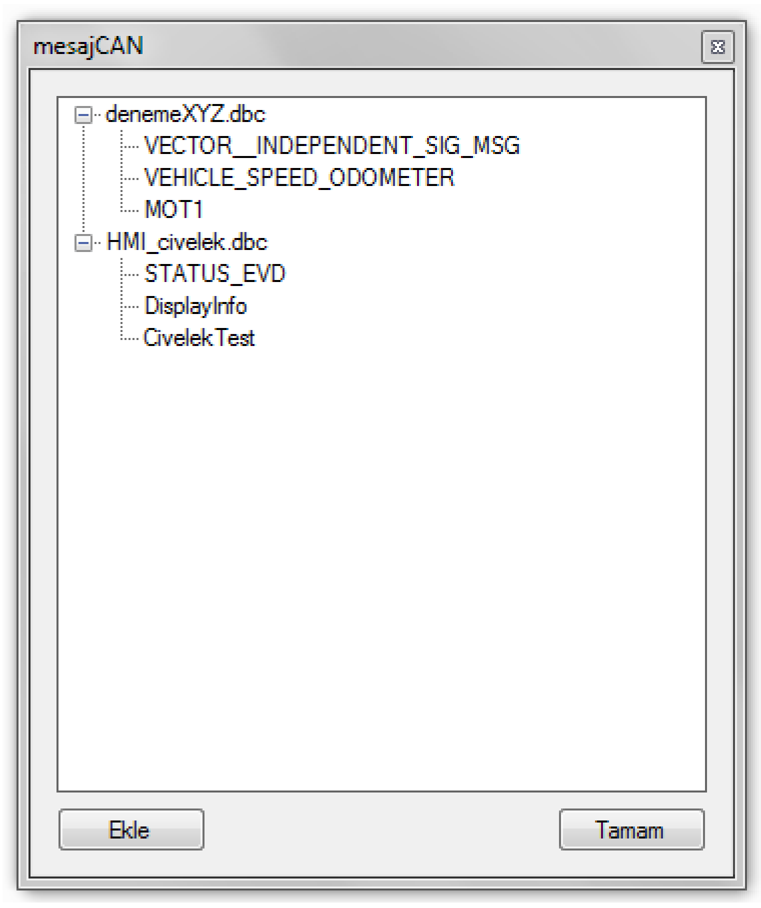


Figure 4.21: *mesajCAN* window

When the "exit" cross is pressed, this window is hidden, not closed. Also, "Add Message" button in first tab of diaCAN is unchecked.

4.3.15 sinyalCAN

sinyalCAN is used to select signals to be tracked for data update and plotted on *grafiCAN*. When a signal is selected and "Add" button is pressed on *sinyalCAN* interface(Figure 4.22), "Up-to-date Data Signals" list is checked for the existence of the signal. If the signal is not added before, it is added to "Up-to-date Data Signals" list, "Up-to-date Data" data table, "Signals' Properties" data table, and *grafiCAN* items. In addition to this control, "Messages to Update Data" dictionary is checked for the existence of the message containing selected signal. If the message is not found, it is inserted to that dictionary, and "Last Values of Updating Messages" data table.

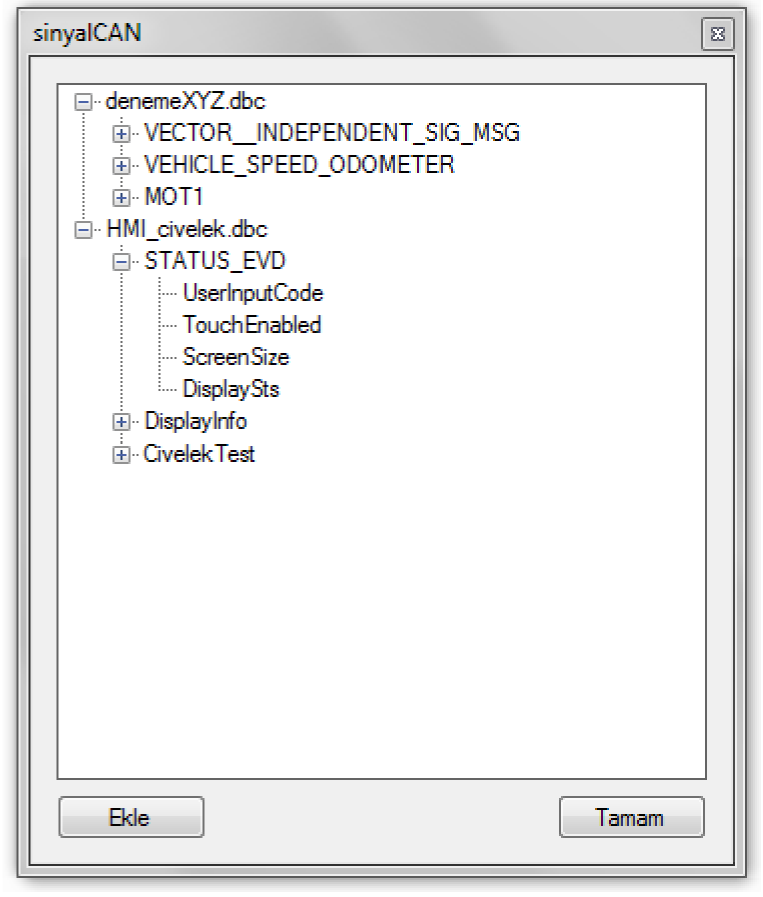
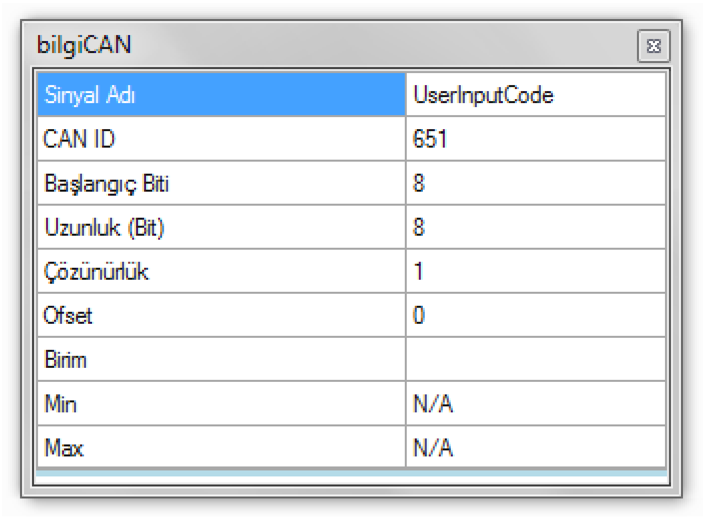


Figure 4.22: *sinyalCAN* window

When "exit" cross is pressed, this window is hidden, not closed. Also, "Add Signal" button in first tab of diaCAN is unchecked.

4.3.16 bilgiCAN

This window(Figure 4.23) is opened when a signal on "Up-to-date Data" data table is double clicked. It gets signal properties from "Signals' Properties" data table and displays them.



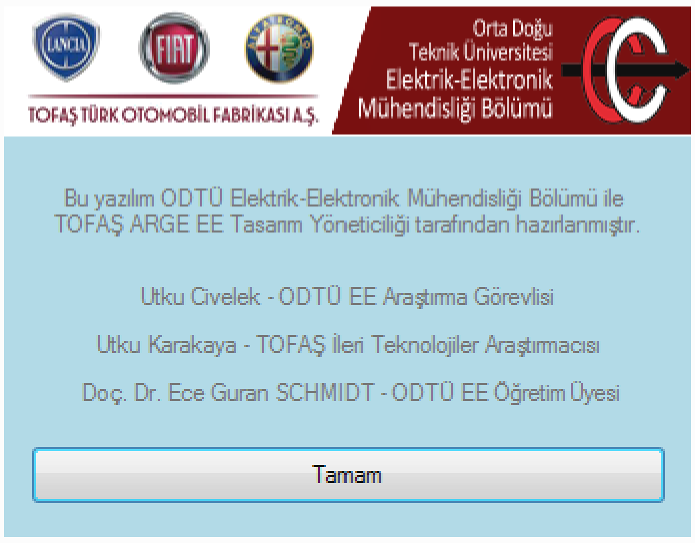
bilgiCAN

Sinyal Adı	UserInputCode
CAN ID	651
Başlangıç Biti	8
Uzunluk (Bit)	8
Çözünürlük	1
Ofset	0
Birim	
Min	N/A
Max	N/A

Figure 4.23: *bilgiCAN* window

4.3.17 Info Screen

This window(Figure 4.24) displays developer company, university, and engineers' names.



TOFAŞ TÜRK OTOMOBİL FABRİKASI A.Ş.

Orta Doğu Teknik Üniversitesi Elektrik-Elektronik Mühendisliği Bölümü

Bu yazılım ODTÜ Elektrik-Elektronik Mühendisliği Bölümü ile TOFAŞ ARGE EE Tasarım Yöneticiliği tarafından hazırlanmıştır.

Utku Civelek - ODTÜ EE Araştırma Görevlisi

Utku Karakaya - TOFAŞ İleri Teknolojiler Araştırmacısı

Doç. Dr. Ece Guran SCHMIDT - ODTÜ EE Öğretim Üyesi

Figure 4.24: Info Screen

4.4 Summary of diaCAN Operation

Interaction of diaCAN items, functions and subjacent windows are given in Figure 4.25.

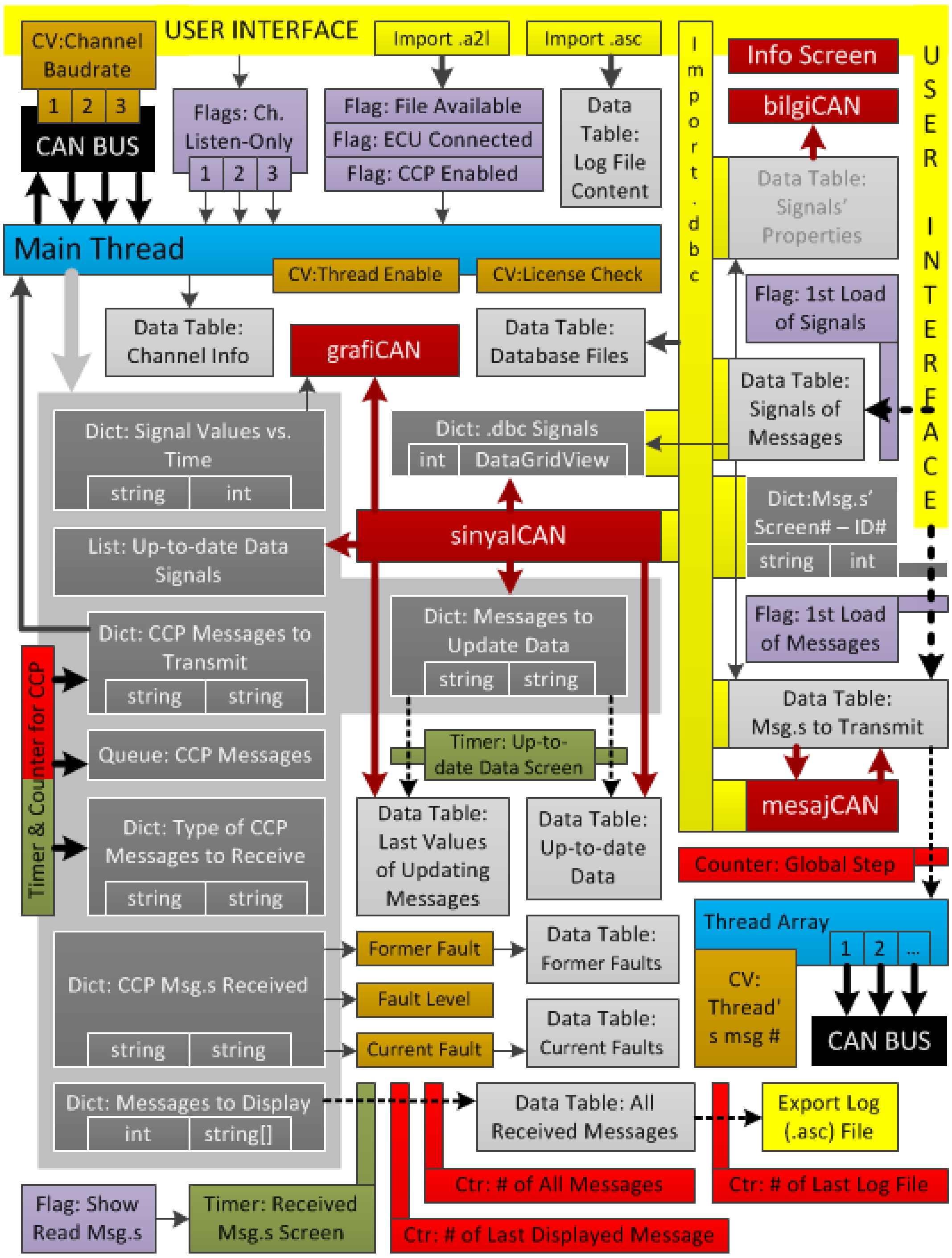


Figure 4.25: diaCAN items and functions' interaction

4.5 Sytem Requirements & Development Workload

Since diaCAN is developed on .NET4 platform, it runs on .NET4 installed computers. Moreover a Kvaser-compatible USB/CAN interface device has to be connected and its firmware should be installed. To access to CAN bus with API functions, diaCAN executable file (446.464 bytes) makes use of "canlib_csharp.dll" dynamic link library file (16.896 bytes).

diaCAN needs a system with at least 512MB of RAM and a 1GHz processor. For the best performance, 2GHz dual processor is required. In idle mode, diaCAN uses 10MB of RAM. When all channels and CCP communication is active, 23MB of RAM is reserved.

Approximately 10 person/day work is done on software design. Coding and debugging lasted for 80 person/day, and tests were conducted in 10person/day.

4.6 Licensing Software by Device Pairing

In order to prevent illegal reproduction and use of the program, diaCAN is designed to run with paired devices only. For each service station, the source code of diaCAN is modified by using UPC (uniform product code) number of USB-CAN interface devices to-be-used. Therefore diaCAN and hardware tools are "paired".

As mentioned in explanation of *acilirCAN*, it checks connected devices' UPC numbers and compare them with the recorded UPC numbers. If a paired device is connected to the computer, diaCAN base window is displayed normally. Otherwise an error text is shown and user is asked to connect a paired device.

The same control is embedded to channel activation checkboxes on "Channel Info" data table of diaCAN base window. If a channel is tried to be activated while no paired devices is connected, an error window is displayed and the channel remains inactive.

For the healthy usage of this licensing process, source code of diaCAN should be obfuscated, i.e. transformed to an equivalent one that is harder to reverse engineer.

CHAPTER 5

TEST RESULTS

We conduct performance tests to examine CPU usage, memory allocation, possible message losses and jitter in periodic message transmission. Kvaser CanKing software is used to generate CAN bus traffic with different loads. Time interval and burst size (number of messages sent in each tick) are modified to change bus load. CAN IDs of messages are increased 1-by-1 to observe whether a problem has occurred in reception or not.

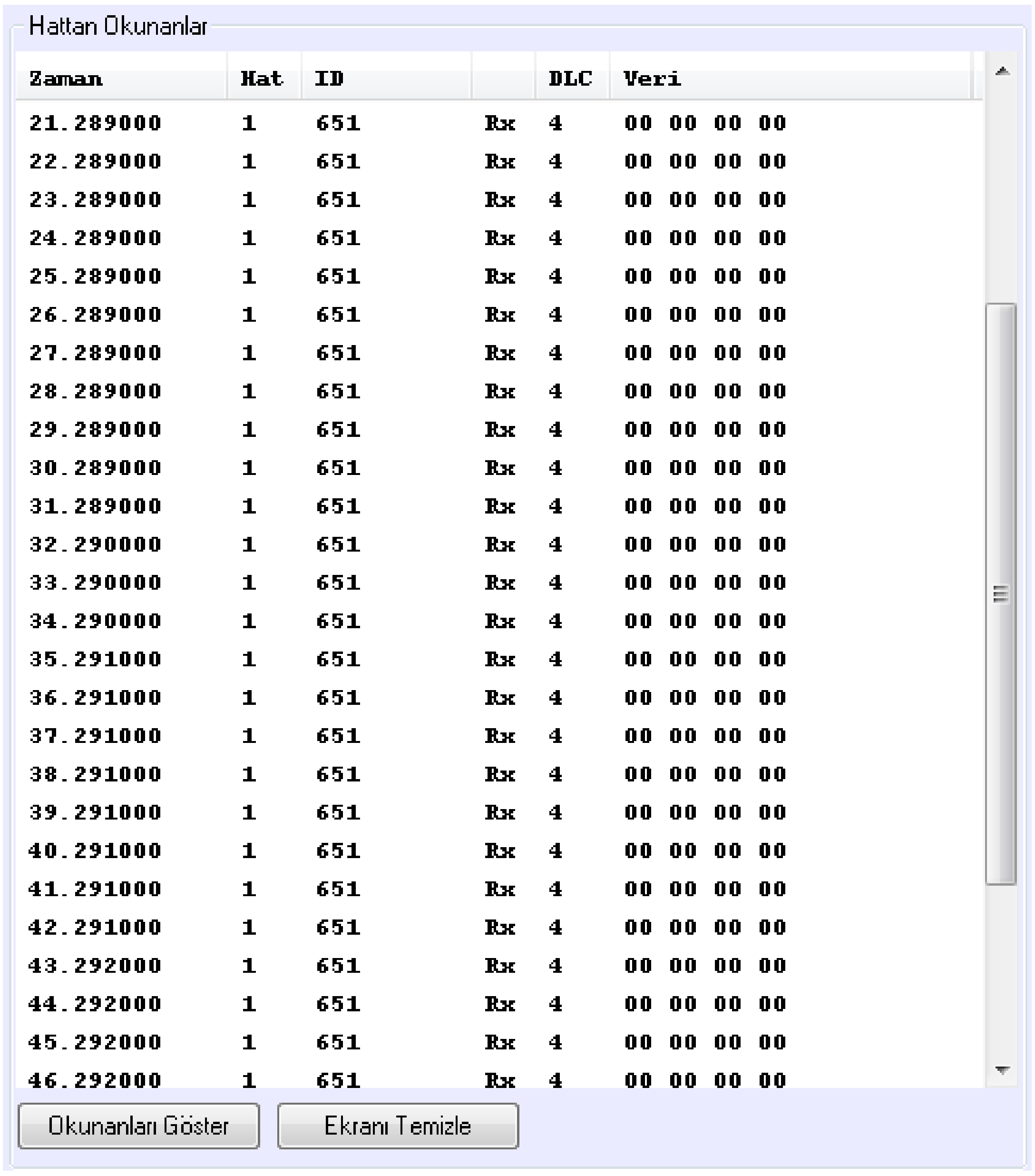
As seen on Table 5.1, diaCAN has two weak points: Memory usage of "All Messages Received" data table and CPU usage of grafiCAN window. As monitoring and sketching time gets longer, diaCAN is forced to save more data to be monitored. Increase in RAM usage is linearly proportional to the total number of received messages (after the screen is enabled) as well as to number and reception frequency of sketched signals. However these weaknesses can easily be overcome by limiting number of messages on "All Messages Received" data table and number of points on grafiCAN window, which is the case in most of the real-time programs, including analyzed CAN software tools in Chapter 6.

Another problem is observed in single-core computers where a sleep is added to main thread of diaCAN to prevent 100% CPU usage. Since the smallest sleep interval is 1ms, average time interval between consecutive messages should be greater than 1ms. Therefore the traffic to be handled by diaCAN must be limited. Otherwise, messages may be lost when the buffer of USB-CAN interface card gets full.

Table 5.1: diaCAN Test Results

CPU	RAM	Main Thread Sleep	# of Sketched Signals	All Msg.s Displayed	CAN Bus Traffic	Time	Result
i3 Single	500MB	1ms	50	Enabled	%11	20min	Lag in grafiCAN after 15min
i3 Single	500MB	1ms	0	Enabled	%11	35min	Lag in interface after 30min
i3 Single	500MB	1ms	50	Disabled	%11	60min	Lag in grafiCAN after 15min
i3 Single	500MB	1ms	0	Disabled	%11	270min	Lag in interface after 260min
i3 Single	500MB	0	-	-	-	-	CPU usage gets 100% immediately
i3 Dual	4GB	0	50	Enabled	%12	12min	OK
i3 Dual	4GB	0	50	Enabled	%23	20min	Out of memory after 20min
i3 Dual	4GB	0	50	Disabled	%70	120min	OK
i3 Dual	4GB	0	50	Disabled	%90	300min	Lag in interface after 280min

The jitter observed (Figure 5.1) in periodic message transmission is usually smaller than 1 millisecond. It rarely exceeds 2 milliseconds, which is still acceptable in CAN bus which is an event-triggered bus with CSMA/CR protocol i.e. no guaranteed message transmission schedule is possible.



Hattan Okunanlar

Zaman	Hat	ID		DLC	Veri
21.289000	1	651	Rx	4	00 00 00 00
22.289000	1	651	Rx	4	00 00 00 00
23.289000	1	651	Rx	4	00 00 00 00
24.289000	1	651	Rx	4	00 00 00 00
25.289000	1	651	Rx	4	00 00 00 00
26.289000	1	651	Rx	4	00 00 00 00
27.289000	1	651	Rx	4	00 00 00 00
28.289000	1	651	Rx	4	00 00 00 00
29.289000	1	651	Rx	4	00 00 00 00
30.289000	1	651	Rx	4	00 00 00 00
31.289000	1	651	Rx	4	00 00 00 00
32.290000	1	651	Rx	4	00 00 00 00
33.290000	1	651	Rx	4	00 00 00 00
34.290000	1	651	Rx	4	00 00 00 00
35.291000	1	651	Rx	4	00 00 00 00
36.291000	1	651	Rx	4	00 00 00 00
37.291000	1	651	Rx	4	00 00 00 00
38.291000	1	651	Rx	4	00 00 00 00
39.291000	1	651	Rx	4	00 00 00 00
40.291000	1	651	Rx	4	00 00 00 00
41.291000	1	651	Rx	4	00 00 00 00
42.291000	1	651	Rx	4	00 00 00 00
43.292000	1	651	Rx	4	00 00 00 00
44.292000	1	651	Rx	4	00 00 00 00
45.292000	1	651	Rx	4	00 00 00 00
46.292000	1	651	Rx	4	00 00 00 00

Figure 5.1: Jitter Observed in Periodic Messages

CHAPTER 6

PREVIOUS WORK ON CAN/ECU SOFTWARE

In this chapter, we comparatively evaluate the available software tools that can transmit, receive and log CAN messages. System requirements, prices and further software capabilities are main three aspects of this analysis. Questioned capabilities are diagnostics, calibration, real-time graphics, and producing arithmetic signals.

There are 4 software tools providing all mentioned capabilities: Vector CANape, ETAS INCA, Samtec SamDiaX, and dSPACE ControlDesk Next Generation.

Vector CANape[1] is one of the sophisticated programs. It facilities all the capabilities that we consider. Furthermore it can be used for design, test and development of ECU networks. It is also used for development of driver assistance systems, image processing algorithms and GPS applications. In addition to CAN bus, CANape also operates on LIN, MOST and FlexRay buses. The Team Collaboration Platform lets both small teams and globally distributed work groups efficiently merge their work results on ECU-calibration process without conflict.

ETAS INCA[2] is another advanced program working on CAN, LIN and FlexRay, including all basic features wanted. The user can program ECU's and record driving data with GPS information. INCA provides animated graphics of engine and instrument cluster, front console dials and road map.

Samtec SamDiaX[3] also offers questioned capabilities as well as ECU programming and virtual bus simulation. It operates on LIN and FlexRay buses, too.

dSPACE ControlDesk Next Generation[4] has extra ECU development, rapid control prototyping and hardware-in-the-loop simulation features. It can access to LIN and FlexRay buses.

RA Consulting DiagRA MCD[16] can acquire data from ECUs and devices on CAN, FlexRay, SMB and Ethernet, make ECU calibration on CAN and FlexRay, and carry out ECU diagnostics on K-Line, CAN and FlexRay. It provides flash programming of ECU's and 3D graphical visualization of signals.

Programs without calibration capability are Vector CANalyzer, Softing CanEasy, Accurate Technologies CANLab, IntrepidCS Vehicle Spy and Influx Module Analyser.

Vector CANalyzer[17] observes, analyzes and supplies data traffic in CAN, LIN, MOST and FlexRay systems. It is a diagnostic tester and observer for UDS and ODX protocols.

Softing CanEasy[18] is, in fact, an ECU engineering program. However it can also be used to accomplish other CAN bus operations.

Accurate Technologies CANLab[19] contains a scripting module with a user configurable editor with syntax highlighting, in addition to basic modules to transmit, receive, display CAN messages. Scripts are used to automate tests, simulate nodes and initiate messages based on network activity.

IntrepidCS Vehicle Spy[20] is utilized in stand-alone flashing of ECU's and stand-alone recording with additional hardware components as well as diagnostics and CAN traffic operations.

Influx Module Analyser[21] has a built-in Python scripting environment for On-Board- Diagnostics and module reprogramming capability.

Other CAN bus analyzer, message generator-observer programs with no calibration and diagnostics applications are:

Schleissheimer GmbH CanEasy[22]

Embedded Systems Academy CANopen Magic[23]

Sontheim CANexplorer 4[24]

Warwick X-Analyser[25]

Ingenieurbüro Für Softwareentwicklung - Xtm[26]

TKE CANtrace[27]

IPETRONIK IPEmotion 2.0[28]

ETAS BUSMASTER[29]

National Instruments[30] has three separate software solutions for CAN bus: NI DIAdem In-Vehicle CAN Data Logger System, NI ECU Measurement & Calibration Toolkit and NI Automotive Diagnostic Command Set. These products are functional with NI LabVIEW.

A compact view of modules, prices, features and usage purposes of these programs are presented in Table 6.1 and Table 6.2. Both tables include the software that is developed in the scope of this thesis, diaCAN.

Another feature of these software tools is that many of them are suitable to operate with CAN/USB interface devices. CAN/USB interface systems ease data acquisition and calibration, since many PCs have bunch of USB ports[31]. They also offer high performance and high reliability.

To shorten service time of vehicles, multi-node calibration system[5] of ShiWei Yang, Lin Yang, and Bin Zhuo can be adapted to these softwares. In this system slave ECUs use a common CAN ID for CRO messages to get broadcast commands. Their DTO message CAN ID's are different from each other as used in traditional CCP. However additional hardware tools are required to run this system.

Another service time shortening solution is automatic measurement and control system[6] proposed by Feng Kong, Liyan Zhang, Jie Zeng, and Yuhua Zhang. However, this system is not suitable to direct adoption since it depends on modification and devotement of two additional ECUs for test and simulation processes.

Table 6.1: CAN-Software Modules & Prices

Company & Product	Diagnostics	Calibration	CPU Req.	RAM Req.	Price
Vector CANape	✔	✔	1 GHz (XP) 2.8 GHz	512 MB (XP) 2048 MB	€5,790.00
ETAS INCA	✔	✔	1 GHz (min) 2 GHz	1024 MB (min) 2048 MB	N/A
samtec samDiaX	✔	✔	N/A	N/A	€4,337.00
dSPACE ControlDesk NG	✔	✔	2 GHz	1024 MB	€2,580.00
RA Consulting DiagRA MCD	✔	✔	N/A	N/A	N/A
TOFAŞ diaCAN	✔	✔	1 GHz	512 MB	N/A
Vector CANalyzer	✔	✗	N/A	N/A	€3,940.00
Softing CanEasy	✔	✗	2 GHz	1024 MB	€4,400.00
Accurate Technologies CANLab	✔	✗	N/A	N/A	N/A
IntrepidCS Vehicle Spy	✔	✗	1 GHz (min) 2.1 Ghz	256 MB (min) 512 MB	$ 2,395.00
Influx Module Analyser	✔	✗	N/A	N/A	£343.00
Schleissheimer GmbH CanEasy	✗	✗	2 GHz	1024 MB	€5,720.00
Embedded Systems Acad. CANopen Magic	✗	✗	N/A	N/A	€999.00
Sontheim CANexplorer 4	✗	✗	1.6 GHz	512 MB	€711.62
Warwick X-Analyser	✗	✗	N/A	N/A	£1000.00
Ingenieurbüro Für Softwareentwicklung Xtm	✗	✗	N/A	N/A	€2,980.00
TKE CANtrace	✗	✗	N/A	N/A	€495.00
IPETRONIK IPEmotion	✗	✗	N/A	N/A	€3,350.00
ETAS BUSMASTER	✗	✗	N/A	N/A	Open Source

Table 6.2: CAN-Software Capabilities

Company & Product	Graphical Display of Signals	Producing arithmetic signals	ECU development or programming	CAN Calibration Protocols	Supported Vehicle Bus Systems (plus CAN)
Vector CANape	✔	✔	✔	CCP, XCP, KWP, UDS, ODX	FlexRay, LIN, MOST
ETAS INCA	✔	✔	✔	CCP, XCP, KWP, UDS, ODX	FlexRay, LIN
samtec samDiaX	✔	✔	✔	CCP, XCP, UDS, ODX	FlexRay, LIN
dSPACE ControlDesk NG	✔	✔	✔	CCP, XCP, ODX	FlexRay, LIN
RA Consulting DiagRA MCD	✔	✗	✔	CCP, XCP	FlexRay
TOFAŞ diaCAN	✔	✔	✗	CCP	-
Vector CANalyzer	✔	✔	✗	UDS, ODX	LIN, MOST, FlexRay
Softing CanEasy	✔	✗	✔	✗	LIN
Accurate Technologies CANLab	✔	✗	✗	CCP, ICP, XCP*	-
IntrepidCS Vehicle Spy	✔	✔	✗	✗	LIN
Influx Module Analyser	✔	✗	✔	✗	-
Schleissheimer GmbH CanEasy	✔	✔	✗	✗	-
Embedded Systems Acad. CANopen Magic	✔	✗	✔	✗	-
Sontheim CANexplorer 4	✔	✗	✗	✗	-
Warwick X-Analyser	✔	✗	✗	✗	LIN
Ingenieurbüro Für Softwareentwicklung Xtm	✔	✗	✗	CCP, XCP*	LIN
TKE CANtrace	✔	✗	✗	✗	-
IPETRONIK IPEmotion	✔	✔	✗	✗	LIN, MOST, FlexRay
ETAS BUSMASTER	✗	✗	✗	✗	-

CHAPTER 7

CONCLUSION

The variety of computer configurations and worker skills in vehicle service stations lead to a need for an user-friendly and low-resource consuming CAN/ECU software tool. In this thesis we develop the diaCAN software tool to fulfill this requirement in industry.

Although diaCAN is developed for vehicle service stations, it can safely be used in factory side. It can handle standard bus traffic with full functionality and excessive bus traffic without displaying all messages. Users can easily calibrate ECU's and get faults recorded without knowing CAN Calibration Protocol transmission rules. Similarly, tests of vehicle components can be conducted by changing trackbars on the diaCAN base window.

In performance tests, tracked signals are updated at least once in 10 seconds. In some tests, this ratio is increased to third times in a second. Except the memory usage weakness mentioned in test results, no problem is observed. diaCAN efficiently handles bunch of processes explained in this thesis.

The performance evaluation tests show that, in single-core computers, CAN traffic above 32,000bps might generate problems. CPU usage is increased to 100% when "Main Thread" is looped without any break. Consequently, user interface response becomes too slow to operate in a healthy way. When a sleep is added to "Main Thread", it can no more handle high CAN traffics. Since the smallest sleep is 1ms; "Main Thread" handles one message in a period larger than 1ms. In computers with two or more cores, CPU usage is always kept low and all messages on the CAN bus are received without loss.

diaCAN, as a native and low cost software tool, requires minumum of hardware resources required by other software tools with diagnostics, calibration, real-time graphics, and producing arithmetic signals capabilities. It can operate on systems with 512MB RAM and 1GHz processors.

Users are expected to calibrate ECU values properly, without negatively affecting other ECU parameters. Any failure or mistake in calibration may lead to unstable or false operation of vehicle since it is a safety-critical software[37]. Therefore a parallel calibration control and cross check mechanism may be a considerable enhancement of diaCAN.

For wider use of diaCAN, Unified Diagnostic Services (UDS), On Board Diagnostics (OBD), and Universal Measurement and Calibration Protocol (XCP) can also be supported. New "UDS", "OBD" and "XCP" tabs with required .NET components can be added to base window of diaCAN.

As provided in Vector CANape, an online data allocation system will be beneficial for service stations. With the combination of measurements and ECU's fault codes, defects can be determined faster by using previous cases' information. Furthermore, machine learning algorithms may be applied for the solution of new encountered conditions.

Another enhancement can be a mobile version of diaCAN. Since CPU and RAM requirements are low enough to operate on tablet PCs and smart phones, diaCAN can be adapted to Android and iOS. With the help of a suitable USB-CAN interface, users can take advantage of longer battery life and high portability offered by these devices.

REFERENCES

[1] http://www.vector.com/vi_canape_en.html/

[2] http://www.etas.com/en/products/inca.php/

[3] http://www.samtec.de/en/hauptmenu/produkte/software/samdiax/

[4] http://www.dspace.com/en/pub/home/products/sw/expsoft/controldesk.cfm/

[5] Yang, S.W.; Yang, L.; Zhuo, B. *Developing a Multi-node Calibration System for CAN Bus Based Vehicle* Vehicular Electronics and Safety, 2006. ICVES 2006. IEEE International Conference on, pp.199-203, 13-15 Dec. 2006.

[6] Feng Kong; Liyan Zhang; Jie Zeng; Yuhua Zhang *Automatic Measurement and Control System for Vehicle ECU Based on CAN Bus* Automation and Logistics, 2007 IEEE International Conference on, pp.964-968, 18-21 Aug. 2007.

[7] Navet, N.; Song, Y.; Simonot-Lion, F.; Wilwert, C. *Trends in Automotive Communication Systems*. Proceedings of the IEEE Volume: 93 , Issue: 6, pp 1204-1223, 2005.

[8] Leen, G.; Heffernan, D.; Dunne, A. *Digital networks in the automotive vehicle*. Computing & Control Engineering Journal, Volume: 10, Issue: 6, pp 257-266, 1999.

[9] http://www.can-cia.org

[10] http://www.iso.org/

[11] BOSCH CAN Specification Version 2.0, 1991.

[12] CCP ASAP Standard Version 2.1

[13] Collberg, C.S.; Thomborson, C. *Watermarking, tamper-proofing, and obfuscation - tools for software protection*. IEEE Transactions on Software Engineering, Volume: 28, Issue: 8, pp 735-746, 2002.

[14] DBC File Format Documentation Version 01/2007

[15] http://www.vector.com/vi_datadescription_ecu1_en.html

[16] http://www.rac.de/en/automotive-products/software/diagnostics/diagra-mcd/

[17] http://www.vector.com/vi_canalyzer_en.html/

[18] http://automotive.softing.com/en/products/caneasy.html/

[19] http://www.accuratetechnologies.com/content/view/179/175/lang,en/

[20] http://intrepidcs.com/VehicleSpy/

[21] http://www.influxtechnology.com/moduleanalyser.html/

[22] http://www.schleissheimer.de/en/products/caneasy/

[23] http://www.canopenmagic.com/

[24] http://www.sontheim-industrie-elektronik.de/en/products/diagnostics/canexplorer-4.html/

[25] http://www.warwickcontrol.com/products/analyzer/x-analyser-for-can-and-lin/

[26] http://www.ibme.de/eng-xtm.html/

[27] http://www.tke.fi/pdf/TKE_CANtrace_datasheet.pdf/

[28] http://www.ipetronik.com/en/software/editions/

[29] http://www.etas.com/en/products/applications_open_source.php/

[30] http://www.ni.com/

[31] Wu Yan; Wang Li-fang; Liao Cheng-lin *CAN Data Acquisition and Calibration System Based on USB Interface* Power and Energy Engineering Conference (APPEEC), 2010 Asia-Pacific, pp.1-4, 28-31 March 2010.

[32] http://www.kvaser.com/index.php?option=com_php&Itemid=258&eaninput=7330130003576&lang=en&product=KvaserUSBcanProfessional

[33] http://www.kvaser.com/

[34] http://www.gapotchenko.com/eazfuscator.net/

[35] http://www.kvaser.com/canlib-webhelp/

[36] http://www.gapotchenko.com/eazfuscator.net

[37] Knight, J.C. *Safety critical systems: challenges and directions*. Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on, pp.547-550, 25-25 May 2002.