A Reliable Message Transmitting of CAN-Bus Network Connection Applied in Real-Time CAR-based Motor Control System

Chun-Shian Tsai[*1], Chia-Wei Huang[2], Ming-Tsai Hsu[3]

[1, 3]Information and Communications Research Lab, Industrial Technology Research Institute (ITRI), Hsinchu, 31040, Taiwan (R. O. C.)

[2]Multimedia and Game Science Dept., Chung Chou University of Science and Technology, Changhua County 510, Taiwan (R.O.C.)

[*1]cstsai@itri.org.tw

***Abstract*- The automobile functions and reliability are increasing rapidly, but the price is decreasing. Therefore, more and more ECUs (Electronic Control Unit) will be conducted into the car. Today, there are about more than 50 ECUs being used in the high-level automobile, and people designed CAN Bus to decrease development cost for car such that ECUs can be worked coordinately. By applying the CAN Bus, we can make the ECUs communicate with each other, moreover, the data message will also be sent to each control device. Therefore, it is an important paradigm for CAN Bus in the CAR-based real time motor control system. In this paper, we first introduce the motor control that is to research how the control message is delivering out through the CAN Bus. Based on this technology, algorithm for motor control is also mentioned. Besides, to achieve much safety and reliability for the real-time motor control, we also research the automotive software framework for ERIKA Enterprise, and through the conducting of ERIKA software, the real time operating system for OSEK can be ported (embedded) into the target ECU hardware in a very easy way. Finally, we propose a demonstrative application for enhanced CAN (ECAN) bus network connection to show how real-time transmission of data frames through ECAN bus network connection is guaranteed by ERIKA Enterprise. In the other words, the motor control for CAR can also be managed by ERIKA to keep the data transmitting in more safety, reliability and real-time.**

Keywords- Automobiles; Real-Time Operating System (RTOS); OSEK/VDX; Embedded System; CAN (Controller Area Network); Motor Control

I. INTRODUCTION

*CAN bus [1] (*for *Controller Area Network)* is a vehicle-based network standard designed to allow *Electronic Control Units (ECUs)* and devices to communicate with each other. For example, the accelerator pedal and motor control system can be equipped with ECUs in car. Vehicle driver can control the car speed by the ECU of accelerator pedal such that an accelerated control message can be sent to the ECU of motor control system via CAN bus. To conduct the ECUs into the car, each mechanical component equipped with ECU can be easily controlled by using the software programming on it. However, today for some automobiles contains already over 70 microcontrollers such that it has the more and more complexity to control on these ECUs. Therefore, the automotive systems are prone to unreliability and the difficulties are in managing. In the other words, how to ensure the safety and the reliability became an issue [2, 3, 4].

The European automobile industry has developed standards for automotive electronics, with the open systems interface specifications OSEK/VDX [5, 6]. It is a set of standards for distributed real-time systems, mainly including *operating system (OS)*, *communication (COM)*, *network management (NM)* and *OSEK implementation language (OIL)* four criteria. It can meet the requirements of security, reliability and resource for areas of vehicle control system, which can ensure the real-time, portability and scalability of vehicle software [7].

Besides the OSEK *real-time operating system (RTOS)* supporting, software controls a large number of functions, which make use of linked networks. Interactions of functions in a linked network contribute to an increasing complexity, which require a strong controllability of the complexity [8, 9, 10]. Thus, a high user-friendly interface of software development in automotive applications is gaining more and more importance.

A satisfiable case for automotive real-time embedded software system called *ERIKA Enterprise [11]* is an open-source RTOS implementation of the ISO 17356 API (derived from the OSEK/VDX API). Erika Enterprise provides a minimal 1-4 Kb Flash real-time kernel (RTOS) for single and multicore embedded systems. Moreover, it followed the OSEK/VDX specifications in the implementation of the source code of OSEK/VDX and it is ready for the OSEK/VDX certification.

In this paper, we mainly focus on two paradigms for the presentation as follows. In the first paradigm, the software implementing technique of motor control will be introduced in Section 3. However, the motor control does not consider its safety and reliability. In order to solve this drawback, we thus have an implementation for the *enhanced CAN (ECAN)* bus network connection based upon the CAR real-time embedded by applying ERIKA Enterprise. This is for the second paradigm. To have a rapid programming designed method in the second paradigm, the software framework for ERIKA Enterprise should

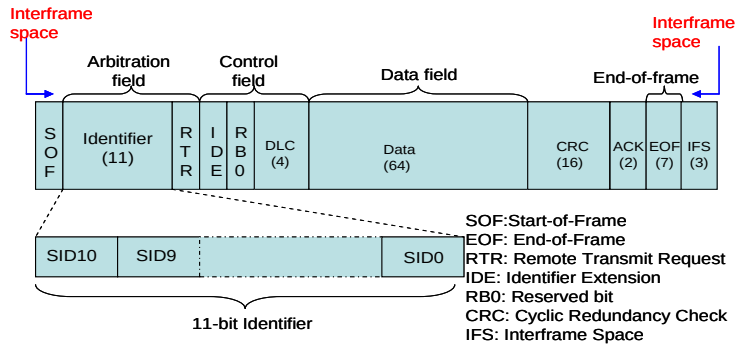
be firstly researched and mentioned. Application experiment for ECAN bus network connection including with OSEK kernel by using the Erika Enterprise demonstrates that it can run well from ERIKA's real-time task scheduling. The meaning (contribution) in this paper is: through the software framework research of ERIKA Enterprise, the OSEK RTOS can be ported (programmed) into target ECU hardware in easily. Finally, the cost of the development for software can be cost down.

The paper is organized as follows. The background of CAN protocol is introduced in Section 2, and then Section 3 introduces the first paradigm for the software implementing technique of motor control. Section 4 presents the second paradigm for an implementation of ECAN bus network connection associated with OSEK kernel by ERIKA programming. Finally, Section 5 discusses the experimental results, and conclusions are given in Section 6.

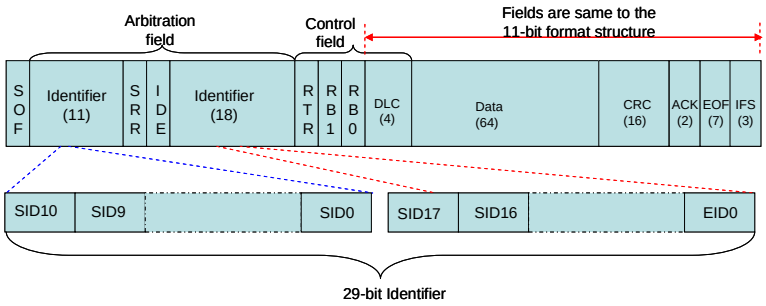
II. RELATED WORK

Controller Area Network (CAN [1, 13]) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other within a vehicle without a host computer. It was designed to reduce the wiring complexity in vehicles, and to enable real-time communication among vehicles ECUs. The CAN 2.0B specification defines two additional data formats: (1) *Standard Data Frame* intends for standard messages that use 11 identifier bits, and (2) *Extended Data Frame* intends for extended messages that use 29 identifier bits.

All CAN nodes connect in a broadcast bus, and CAN using with bitrates can be up to *1 Mbit/s*. The CAN bus protocol uses asynchronous communication. Information is passed from transmitters to receivers in data frames, which are composed of byte fields that define the contents of the data frame, as displayed in Figure 1-(a).



(a) 11-bit format structure(standard data frame)



(b) 29-bit format structure(extended data frame)

Figure 1 CAN message format

For the case of the standard format, the identifier defines the type of information contained in the message and is used by each receiving node to determine if the message is of interest to it. The identifier is unique and it determines the frame priority, and enables receivers to filter frames [12]. The CAN transmits data in frames containing a header and 0 to 8 bytes of data. The number of bytes is user-selectable.

On the other hand, for the case of the extended data frame, it begins with a start-of-frame (SOF) bit followed by a 31-bit Arbitration field, as shown in Figure 1-(b). The Arbitration field for the extended data frame contains 29 identifier bits in two fields separated by a *Substitute Remote Request (SRR)* bit and an IDE bit. The SRR bit determines if the message is a remote frame. The IDE bit indicates the data frame type. For the extended data frame, IDE is set, and for the standard data frame, IDE is cleared.

III. SOFTWARE IMPLEMENTING TECHNIQUE OF MOTOR CONTROL

For the software implementing technique in motor control, some data message should be predefined to indicate the function

of command protocol. Therefore, a detailed presentation for command protocol would be first presented in the Section 3.1. The algorithm for the motor control would be introduced in Section 3.2. Finally, the environment setup and experiment results are introduced in Section 3.3 and Section 3.4, respectively.

A. Command Protocol

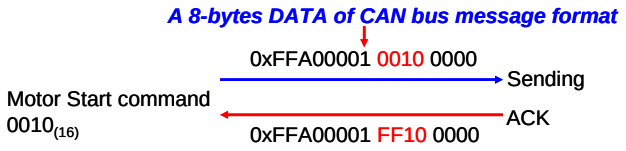
The message transmitting for motor control must be followed in the format of CAN bus. In order to have a successful transmitting from source to destination, *individual device ID* for command protocol should be firstly predefined by user programming to indicate the sender and receiver. As illustrated in Figure 2, the individual device ID of sender in *Explore 16 EVB* can be defined for hexadecimal value $A00001_{(16)}$, and the individual device ID of receiver in *dsPICDEM MCLV* can be defined for hexadecimal value $B00001_{(16)}$.

Explore 16 EVB: TX_ID➔A00001 (sender)
RX_Filter➔B00001 (receiver)

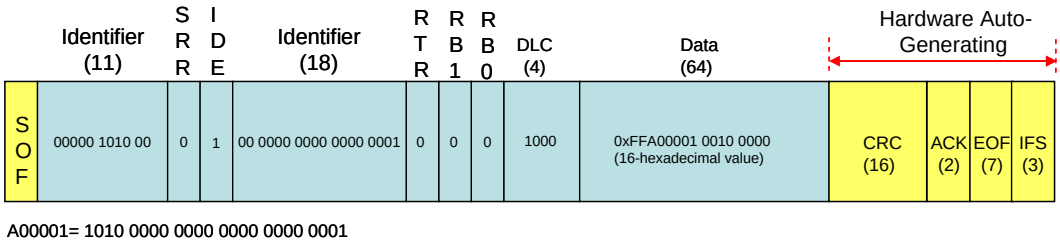
dsPICDEM MCLV: TX_ID➔B00001 (sender)
RX_Filter➔A00001 (receiver)

Figure 2 Predefined device ID by user programming viewpoint

On the other hand, the programmer would also need to define message functions of command protocol for motor control in hexadecimal value as follows: (1) *motor start-up* is defined as $0010_{(16)}$, (2) *motor stop* is defined as $0020_{(16)}$, (3) *motor speed-up* is defined as $0030_{(16)}$, (4) *motor speed-down* is defined as $0040_{(16)}$, (5) *motor response value* is defined as $0050_{(16)}$. Note that the message functions in hexadecimal value are all for user programming defined. As illustrated in Figure 3-(a), programmer can predefine a message function in hexadecimal value $0010_{(16)}$ to indicate a motor start-up command. The transmitting procedure between source and destination via CAN bus mainly follows in a message format which is composed of an *8-bytes DATA* (*0xFFA00001 0010 0000*) and a header information. The first byte of the *DATA* field can be filled in any hexadecimal value (e.g. *0xFF*). These fields for yellow color in Figure 3-(b) such as *SOF*, *CRC*, *ACK*, *EOF* and *IFS* indicate a hardware generating. The *acknowledgement message (ACK)* is always to be replied to sender whenever the data frame is successfully received by destination. All the ACK messages should be included with a hexadecimal value *0xFF* in the fifth byte as shown in Figure 3-(a).



(a) Motor start-up for sending and acknowledgement



(b) CAN bus message format

Figure 3 Motor start command protocol

B. The Motor Control Algorithm

In the presentation of the motor control algorithm, as illustrated in Figure 4, there are three phases to be mentioned. The first phase in *A1/B1* shows an initiated procedure for development boards in *Explore 16 EVB* and *dsPICDEM MCLV*. The initiation is included with the parameter setup of oscillator, the initialization of module function (for enabling these module functions in CAN, DMA, LCD, PWM, ADC and TIMER) and the enabling setup of interrupt. The second phase in *A2/B2-1/B2-2* presents a CAN bus waiting procedure for two development boards in between *Explore 16 EVB* and *dsPICDEM MCLV*. For the *Explore 16 EVB board*, the phase *A2* would be required to wait the CAN bus message which is transmitted from *dsPICDEM MCLV*. If the message can be successfully received, then the message function of command protocol should be displayed on LCD of *Explore 16 EVB* board. The third phase in *A3/B3*, the board for *Explore 16 EVB* keeps detecting in whether user has already pushed a button (*S3/S6/S5/S4*). If not, it would go back to second phase in *A2* going continuously to wait the message of CAN bus. Otherwise, a related message for command protocol would be displayed on the LCD of *Explore*

16 EVB board and it would also be transmitted to destination via the CAN bus. Please note that the button functions need to be predefined by programmer. Please note for this paper, the button *S3* can be identified as the function of motor start, the button *S6* can be identified as the function of motor stop, the button *S4* can be identified as the function of motor speed-down, and the button *S5* can be identified as the function of motor speed-up. Functions for these buttons can be easily associated with command protocol as mentioned in Section 3.1.

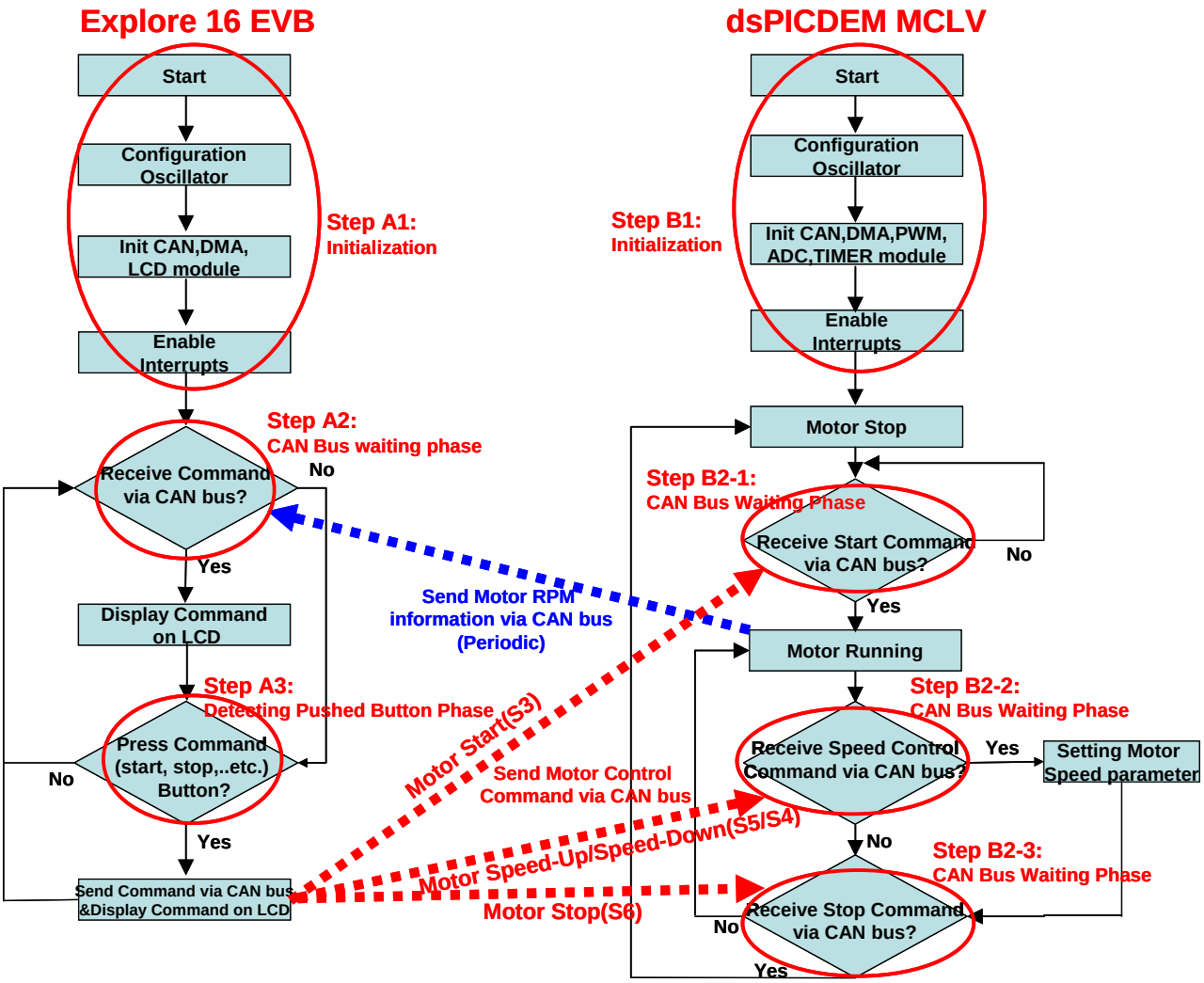


Figure 4 The message transmitting flow chart of CAN bus

On the other hand, destination board for *dsPICDEM MCLV* in second phase *B2-1* is also waiting in whether the CAN bus message included with the function of motor start in *S3* can be transmitted from *Explore 16 EVB* source. If not, it goes continuously keeping the detection. Otherwise, the microcontroller ECU would begin producing *PWM (Pulse-Width Modulation)* signal to enable the motor for running. Next, phase *B2-2* is mainly to detect if the messages for motor speed-up (*S5*) and motor speed-down (*S4*) are received by *dsPICDEM MCLV*. If it is "Yes", parameters for motor control should be set in order to adjust the speed of the motor. Otherwise, next phase in *B2-3* would prepare in checking whether the CAN bus message for motor stop (*S6*) can be received. If it is "Yes", the PWM signal for motor stop can be sent to the motor. Otherwise, it goes back the phase *B2-2* to check CAN bus message for waiting.

C. Environment Setup

In this section, we implement the algorithm for the motor control based upon *Microchip's* development boards. Integrated developing environment and operating interface are depicted in Figure 5. The board in *Explore 16 EVB* applies a *16-bit* microcontroller for *dsPIC33FJ256GP710A* to manage and send the motor control command to destination via CAN bus, and the board in *dsPICDEM MCLV* uses a *16-bit* microcontroller for *dsPIC33FJ256MC710A* to generate and management the PWM signal of the motor control to the motor. Network connection among these boards is always via the CAN bus. There are four buttons, which are namely *S3*, *S4*, *S5* and *S6*, plugged in *Explore 16 EVB* board. Thus, we should also predefine these functions by programmer as mentioned in Sections 3.1 and 3.2. Once the microcontroller of *Explore 16 EVB* board detects a button to be pushed by user, data message can be transmitted immediately through the CAN bus. The CAN bus message

format adopts an extended data frame in this experiment. Destination for *dsPICDEM MCLV* development board is responsible to receive the command message of *Explore 16 EVB* source board. After receiving, the PWM signal which generated by ECU of *dsPICDEM MCLV* is sent to *BLDC* motor for the controlling.

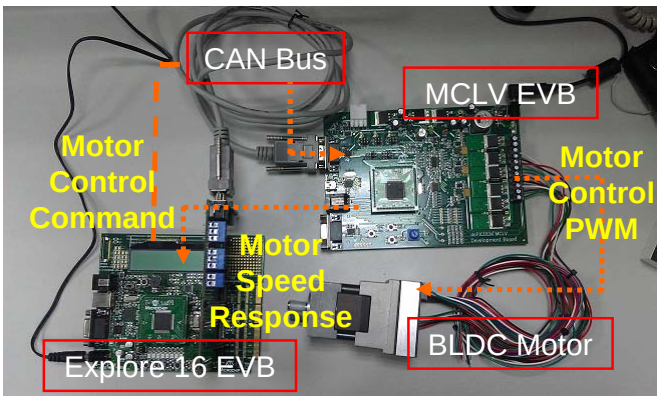


Figure 5 The integrated development environment of CAN Bus

D. Experiment Results

In this section, the experiment results for motor control in message transmitting are always through the CAN bus. As displayed in Figure 6, a button called *S3* as the function of motor start should be predefined by programmer. The LCD shows a *TX ID* which indicates an identification of *Explore 16 EVB* in sender case. Message for CAN bus consists of *TX ID* and *S3* generating in an *8-bytes DATA* as mentioned in Figure 3-(a). The DATA would be encapsulated into a new data frame included with the other header related information of Figure 3-(b) for transmitting to the CAN bus. Whenever destination for *dsPICDEM MCLV* receives the message of motor start (*S3)*, microcontroller generates a PWM signal to control the motor. Moreover, status for motor start is also sent back to *Explore 16 EVB* to display a word "RUN" on LCD. On the same way, the other messages for motor stop (*S6)*, motor speed-up (*S5)* and motor speed-down (*S4)* are displayed in Figure 7, Figure 8 and Figure 9, respectively. Finally, the Figure 10 shows a responding status message from *dsPICDEM MCLV* to *Explore 16 EVB*.

Figure 6 Motor start by pushing button S3 (from Explore 16 EVB)

Figure 7 Motor stop by pushing button S6(from Explore 16 EVB)

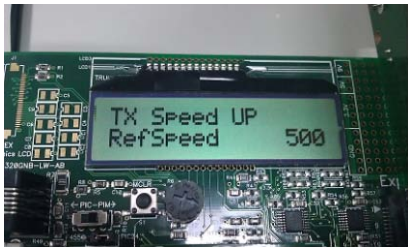


Figure 8 Motor speed-up by pushing button S5(from Explore 16 EVB)

Figure 9 Motor speed-down by pushing button S4(from Explore 16 EVB)

Figure 10 The status of motor speed responded(from dsPICDEM MCLV)

In this section, we practice the software implementing technique for the algorithm of motor control. But, this technique does not consider its safety and reliability, especially in a real-time solution. Once migrating into a real vehicle case, the safety is much required to be certificated. In next section, we would research the implementation for *enhanced CAN (ECAN)* bus network connection which is about CAR-based real-time embedded software system. By applying this implemented technique of CAR real-time embedded software system, the drawback for motor control can be solved to achieve this goal in safety, real time and reliability.

IV. CAR REAL-TIME EMBEDDED SOFTWARE SYSTEM

For this section, we research an implemented method of the enhanced CAN-bus network connection by adopting ERIKA

Enterprise programming. The ERIKA Enterprise includes with OSEK real time operating system for car. Through this kind of programming, we can have an easily porting method into the embedded system. In this section, the implementation contains with the concepts in: (1) framework research of automotive software system, (2) programming designed method for ECAN-bus network connection, and (3) procedure of compilation and porting, which will be introduced in the following sections, respectively.

A. Framework Research of Automotive Software System

The automotive software system in this paper is referred to an open source which called *ERIKA Enterprise*. In this section, we research and draw the framework of ERIKA Enterprise in Figure 11. The ERIKA supports *OSEK kernel*, *RT-Druid* and *CAL(C/C++ actor language)* which will be detailed in the following.

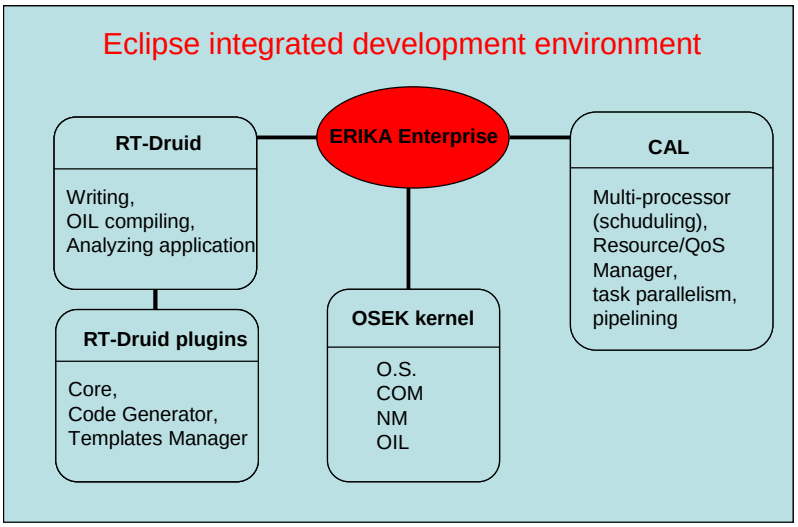


Figure 11 The framework of ERIKA Enterprise software system

OSEK/VDX is a set of standard for a distributed, real-time architecture that was developed by a consortium of European automobile manufactures and suppliers in conjunction with the University of Karlsruhe, Germany. It is primarily comprised of four standards: the *operating system (OS)*, *communication (COM)*, *network management (NM)*, and the *OSEK implementation language (OIL)*. Three additional standards are in progress: the *OSEK/VDX real-time interface (ORTI)*, the *OSEK/VDX Time-Triggered Operating System*, also known as OSEK-time, and the *OSEK/VDX Fault Tolerant Communication specification*. For more details, please refer to the Literatures [5, 6].

RT-Druid is a set of open-source *Eclipse* plugins implementing a configuration language inspired on OSEK OIL, able to produce an OSEK ORTI file compatible with *Lauterbach Trace32 debuggers*. The Eclipse Platform is a multi-language software development environment comprising an *integrated development environment (IDE)* and an extensible plug-in system. It is written mostly in Java. By means of various plugins, it can be used to develop applications in various programming languages including ERIKA Enterprise, RT-Druid, and CAL language. The RT-Druid is a part of ERIKA Enterprise, and it allows writing, OIL compiling, and analyzing application in a comfortable environment. RT-Druid is composed by a set of plugins for the Eclipse Framework. The following is a list of the available plugins. The RT-Druid Core plugin contains all the internal metamodel representation, providing a common infrastructure for the other plugins, together with *ANT scripting* support. The RT-Druid Code Generator plugin implements the OIL language compiler, together with target independent code generation routines for ERIKA Enterprise. The RT-Druid Templates Manager plugin provides the possibility to easily write application templates to be used when generating a new application.

CAL is an actor-oriented language with data-flow programming models and it describes algorithms using a set of encapsulated dataflow components called "*ACTORS*" communicating with each other. The only interaction an actor has with another actor is usually through input and output ports. Three techniques will be combined in ACTORS: (1) *Visualization*, (2) *Feedback control*, and (3) *Data-flow programming models*. Virtualization techniques such as reservation-based scheduling provide spatial and temporal separation of concerns and enforce dependability and predictability. ACTORS addresses design of resource-constrained software-intensive embedded systems with high requirements on adaptivity and efficiency. Feedback will be used to dynamically adjust the size of the reservations based on acual resource consumption and *quality-of-service (QoS)* (global feedback) and to adjust the resource consumption within the individual streams (local feedback). Dataflow programming with CAL has been developed with the objective of exposing coarse-grained parallelism as much as possible. Coarse-grained parallelism means explicit parallelism in a CAL program (at actor level). The parallelisms are with concurrency (e.g. task parallelism) and pipelining. Thus, the one of target platforms architectures on which the specific CAL model under analysis will be also focus on is multiple processors (*N cores*) and multiple FPGA. In this paper, we mainly focus on RT-Druid's OIL structure programming designed and OSEK real-time embedded system involved in ERIKA Enterprise, but

not the CAL language designing which remains as a future work.

The ERIKA Enterprise is with multi-processor *real-time operating system kernel (RTOS)* which followed the OSEK/VDX specifications in the implementation of the source code, and it is ready for the OSEK/VDX certification. Moreover, it also supports a variety of 8, 16, 32 bit microcontrollers, including multicores. For example, ARM Cortex MX, Freescale PPC e200(MPC 56xx), Microchip PIC 32, Microchip dsPIC, TI MSP430…etc. The ERIKA Enterprise is available for various hardware platforms and introduces task scheduling concepts, resource sharing with *Immediate Priority Ceiling protocol*, real-time mechanisms and programming features to support and exploit the microcontrollers and multi-core systems-on-chip.

The benefits for ERIKA enterprise are easily for the migration from a single core to multiple cores, which means no changes to the application source code, only simple modifications to different OIL configurations. Moreover, in our research, we adopt the programming rule of ERIKA framework such that real-time solution is based on the existed solution from ERIKA. In next subsection, we research the designing method for ECAN-bus network connection by associating the features of ERIKA Enterprise.

B. Programming Designed Method for ECAN-bus Network Connection

In Erika Enterprise all the RTOS objects like tasks, alarms, resources, are static and predefined at application compile time. To specify which object exists in a particular application, Erika Enterprise uses the OIL Language, which is a simple text description language. Here is a general case of the OIL structure for ERIKA Enterprise as displayed in Figure 12. The OIL contains with seven components which include the *OS*, *TASK*, *APPMODE*, *RESOURCE*, *EVENT*, *COUNTER*, and *ALARM* for available ERIKA programming. Application user can choose various components based on their requirement.

```
/* OIL structure for ERIKA Enterprise */
CPU mySystem
{
    /* The symbol for xxx indicates a component name define. */
    OS xxx
    {
        /* OS is the Operating System which runs on the CPU. This object contains all the global
           settings which influence the compilation. */
    };
    TASK xxx
    {
        /* TASK is an application task handled by the OS. */
    };
    APPMODE xxx
    {
        /* APPMODE defines the different application modes. These modes are then used to control
           the autostart feature for tasks and alarms in the OIL file. */
    };
    RESOURCE xxx
    {
        /* RESOURCE is a resource (basically a binary mutex) used for mutual exclusion. */
    };
    EVENT xxx
    {
        /* EVENT is a synchronization flag used by extended tasks. */
    };
    COUNTER xxx
    {
        /* COUNTER is a software source for periodic / one shot alarms. */
    };
    ALARM xxx
    {
        /* ALARM is a notification mechanism attached to a counter which can be used to activate a
           task, set an event, or call a function. */
    };
}
```

Figure 12 OIL structure of ERIKA Enterprise

In this paper, we research and summary a programming design method for ECAN-bus network connection based upon OIL structure. As displayed in Figure 13, there are at least four classes of objects used: (1) *OS*, (2) *TASK*, (3) *COUNTER*, and (4) *ALARM*.

```
/* OIL structure for ECAN-bus network connection */
CPU mySystem
{
        /* The name for OS object is defined by “myOS”. */
        OS myOS
        {
            LDFLAGS parameters setup  /* Containing a list of additional linker parameters */
            EE_OPT parameters setup /*  Compilation flags and options setup */

           CPU_DATA = xxx   /*  The symbol xxx indicates a CPU name, e.g. PIC30 */
           {
              APP_SRC = "code_name.c";  /*  Import an application source code from a C file */
              …..
           };

           MCU_DATA = xxx /*  The symbol xxx indicates a MCU name, e.g. PIC30 */
           {
              MODEL = MCU type;  /* e.g. PIC33FJ256MC710 is a 16 bit Microchip MCU */
           };

           BOARD_DATA = board name
            /* e.g. The board name may be replaced by MICROCHIP_EXPLORER16  */
           {
              Board functions setup
              /* e.g. USELEDS = TRUE, USELCD = TRUE, USEBUTTONS = TRUE, USEANALOG = TRUE */
           };
           KERNEL_TYPE setup; /*Scheduler choice (e.g. KERNEL_TYPE = FP, fixed priority scheduler) */
       };
       TASK task_name  /* It may have many tasks based on system requirements */
       {
           PRIORITY number setup  /* e.g. PRIORITY = N, the symbol N indicates a variable number with
                                     respect to various tasks */
           STACK = SHARED;
           SCHEDULE = FULL;
       };
       COUNTER counter_name;
       ALARM alarm_name /* It may have many ALARMs based on the number of tasks */
       {
          COUNTER = " counter_name  ";
          ACTION = ACTIVATETASK { TASK = “task_name"; }; /* activate a task by using ALARM */
       };
}
```

Figure 13 Programming design method of OIL structure for ECAN-bus network connection

In (1), the OS object introduces requirement in linker parameters setup in *LDFLAGS*. The *EE_OPT* is a way to specify configuration flags to the ERIKA build environment, and it can also be specified as strings in the OS section of the OIL file. The *EE_OPT* contains a list of additional compilation flags passed to the ERIKA Enterprise makefile. In practice, the *EE_OPT* makefile variable controls which files has to be compiled and with which options. The *CPU_Data* section of the OS object is used to specify the configuration of a core in a single or in a multiple core device. The *MCU_DATA* section is used to specify configuration of a specific microcontroller. The *BOARD_DATA* section is used to specify the configuration of the board where the microcontroller is placed. For example, the board configuration includes the configuration of the external devices like leds, buttons, displays, and other peripherals.

In (2), the TASK section is an application task handled by the OS. Each PRIORITY is with corresponding to a task scheduled number. The OS may allow many tasks to be performed in concurrently. In (3), the COUNTER is a software source for periodic or one shot alarms. In (4), the ALARM is a notification mechanism attached to a counter which can be used to activate a task, set an event, or call a function.

C. Procedure of Compilation and Porting

In order to have code migration from software to hardware MCU, the compilation and porting processes should be researched and introduced in this paper. As shown in Figure 14, the ERIKA enterprise, RT-Druid and Cygwin are plugins of Eclipse software. Source project which includes with C file (e.g. *main_code.c*), OIL file (e.g. *conf.oil*), and C header files is firstly imported into Eclipse. The OIL file provides a method of configuring the objects in an OSEK/VDX implementation for a specific application. The system is configured by using an OIL configuration file that contains the definition of the application. Through adopting the plugins of ERIKA Enterprise, RT-Druid is responsible for OIL compilation.

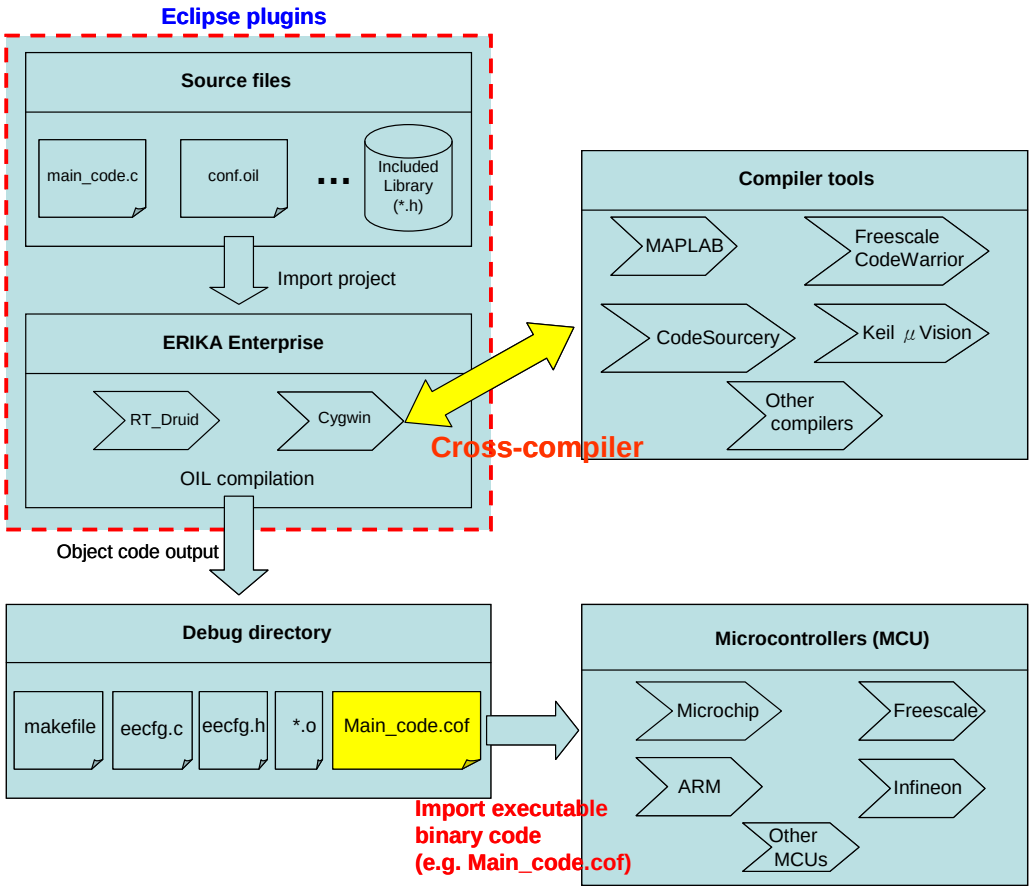


Figure 14 Compiling and porting processes on a microcontroller

The Cygwin is a collection of tools which provide a Linux look and feel environment for Windows. It includes also a DLL (*cygwin1.dll*) which acts as a Linux API layer providing substantial Linux API functionality. Moreover, the Cygwin provides a compiling environment for cross-compiler which introduces various compiler tools. Each hardware MCU is corresponding to its related compiler tool. For example, the hardware for Microchip is corresponding to the *MAPLAB's* compiler, and ARM is corresponding to *CodeSourcery's* ARM compiler.

In the process of the compilation for ERIKA Enterprise, RT-Druid is the tool used to configure the ERIKA Enterprise OS. RT-Druid produces at least three files (e.g. *makefile*, *eecfg.h* and *eefg.c*) in the project output directory which is normally called Debug. Others files may be produced, depending on the configuration or the target architecture. The main *makefile* defines some EEOPTs and starts the compilation process. The C header for *eecfg.h* file contains the definitions of macros for constant parameters such as task IDs, the number of tasks…etc. This header is included also in some assembly files. The C file for *eecfg.c* contains the definition of all the kernel data structures that depend on the configuration (e.g. an OIL file).

Besides outputting those files as mentioned above, object codes and a COF file are also required. The COF file which derived from object code (e.g. **.o* file) is an executable binary code with respect to a hardware MCU. Finally, the user programming for code migration into the MCU is by applying one of the related compiler tools. For example in Microchip MCUs, the COF file is a compatible MAPLAB IDE COF file. The MPLAB IDE is the new graphical, integrated debugging tool set for all of Microchip's more than 800 8-bit, 16-bit and 32-bit MCUs and digital signal controllers, and memory devices. By using MAPLAB IDE, the COF file can be ported on MCUs.

V. EXPERIMENTATION FOR ECAN-BUS NETWORK CONNECTION

A. Components Requirement

In this paper, we have two experiments in cases of the sending and receiving for ECAN-Bus network connection by adopting ERIKA Enterprise programming. The components requirement is described as follows. We adopt a low cost and efficient Explore 16 development board to evaluate the ECAN-bus network connection features and performances by using ERIKA Enterprise with OSEK standard programming. The *PICtail Plus interface* (i.e. a daughter board contained with ECAN/LIN bus) is used for connection to *Explorer 16 Development Board* for 16-bit and 32-bit MCUs. In this experimentation, Microchip's microcontroller for *PIC33FJ256MC710* is applied. Coupled with the MPLAB ICD 3 In Circuit Debugger, real-time emulation and debug facilities speed evaluation and prototyping of application circuitry. Moreover, CAN

Bus Analyzer of Microchip for hardware *APGDT002* and a related PC software (mainly for application layer) is also adopted in order to analysis data frame of ECAN-bus. Hardware for CAN bus analyzer is with *Encapsulation/Decapsulation* functions to send/receive ECAN-Bus data frame.

B. Starting and Ending

For the procedure of the compilation, we introduce the details in Figure 15, the compilation processes of OIL file (i.e. *conf.oil*) are separated from sending to receiving cases. Based on the choice of the cases, the OIL file needs to import its related OIL file. For example in the sending case, that means we want to have an experimentation from the sending case, the *conf.oil* file is firstly required to be imported the *sender.oil* file, which will also need an imported C file of the *sender.c*. By applying the compiling processes of ERIKA Enterprise, as mentioned in Figure 14 of Section 4, an executable binary code such as *sender.cof* file is created. Finally, the binary code has a migration and porting into the Microchip's MCU. On the same way, another case on receiving for compilation procedure is also same to the sending case.

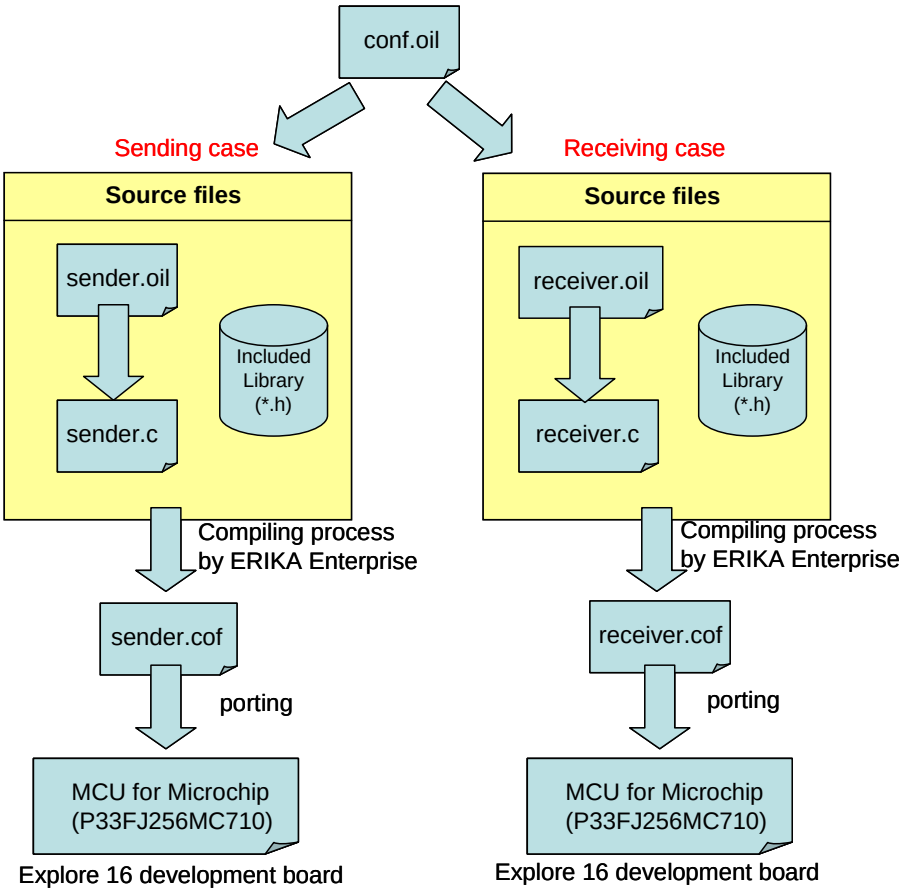
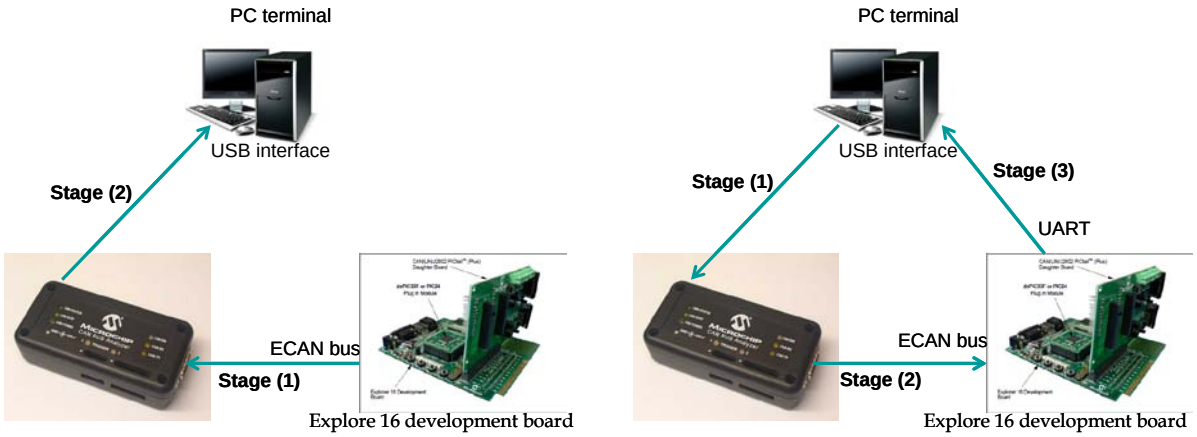


Figure 15 Compilation procedure in cases of the sending and receiving

The ERIKA Enterprise is contained with OSEK kernel and simple description for OIL structure. Source code for ECAN-Bus transmission and reception technologies can be programmed by the Task concept of OSEK operating system. The tasks for code should be programmed in OIL and C files. Application user can use ALARM to activate its related TASK which is also presented in the C file. The TASKs are easily followed the philosophy of subroutine programming in the C file when it needs OSEK RTOS supporting. By following the programming designing method of OIL structure, as mentioned in Figure 13 of Section 4, the OSEK will have a suitable real-time scheduling for Tasks.

C. Experiment Results

The topologies for ECAN-Bus network connection in sending and receiving cases are displayed in Figure 16. Related experiment results of Figure 16-(a) and Figure 16-(b) are shown in Figure 17 and Figure 18, respectively. In Figure 16-(a), the sending case describes that messages must be sent by ECAN-bus from Explore 16 development board to CAN bus analyzer. The analyzer will be responsible to decapsulate the received data message (data frame) and display the content of the data frame to application layer (e.g. PC terminal) in Figure 17. For the experiment result in Figure 17, we apply an extended data frame (29-bit identifier) for enhanced CAN bus protocol. Each receiving data frame presents an 8-bytes length for size, and these data messages can be followed in the task scheduling of OSEK RTOS standard and CAN-bus network connection protocol.



(a) Sending case (b) Receiving case

Figure 16 ECAN-bus network topologies

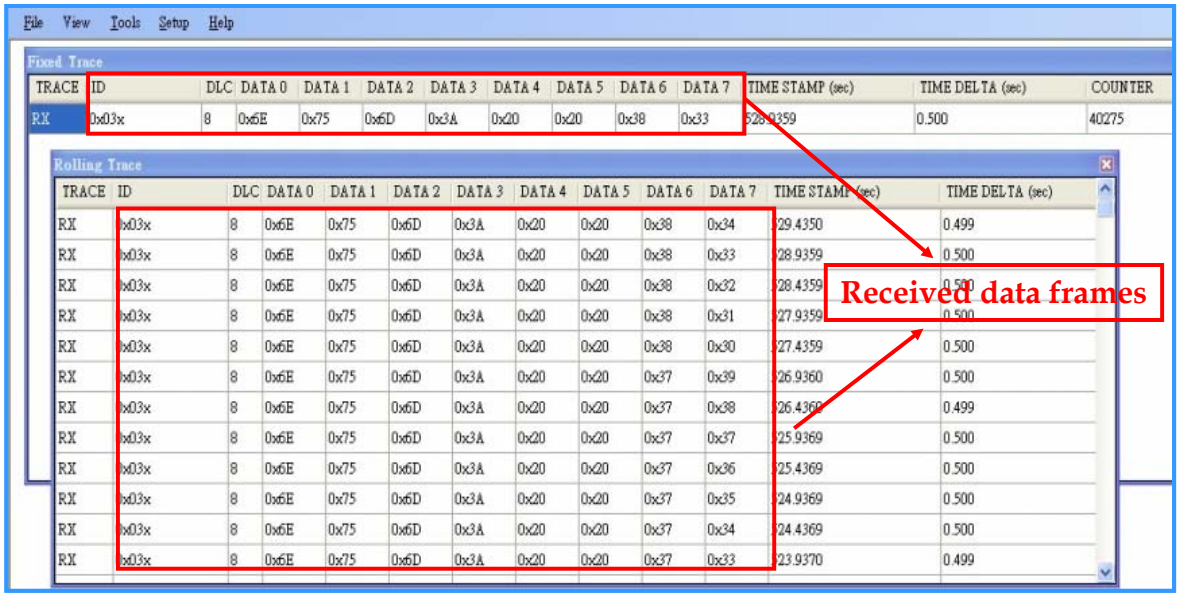
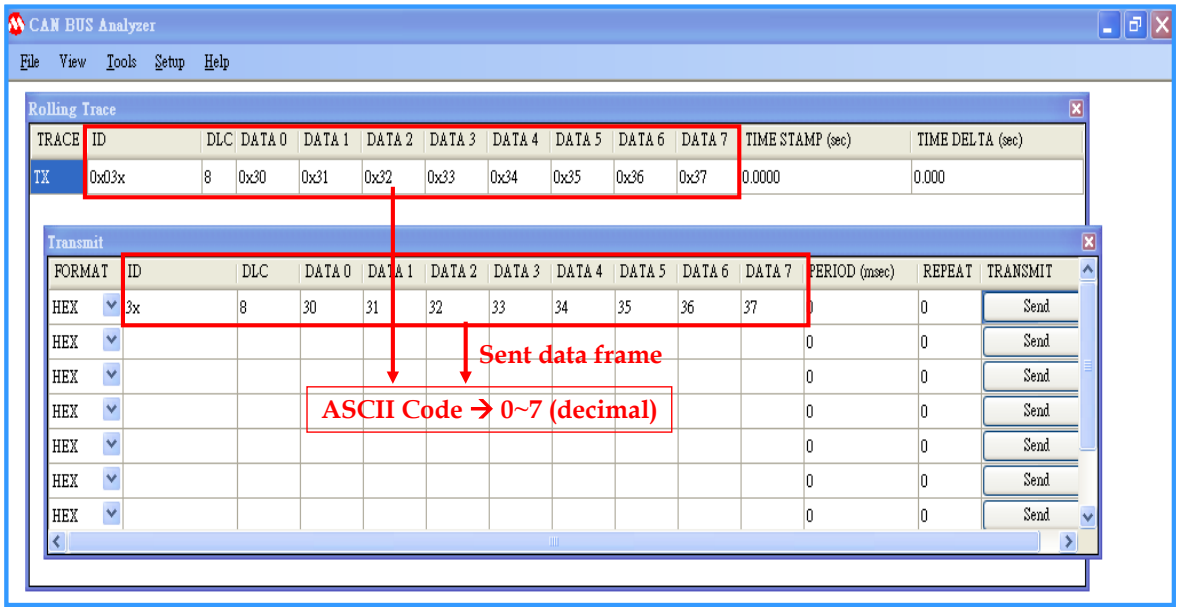
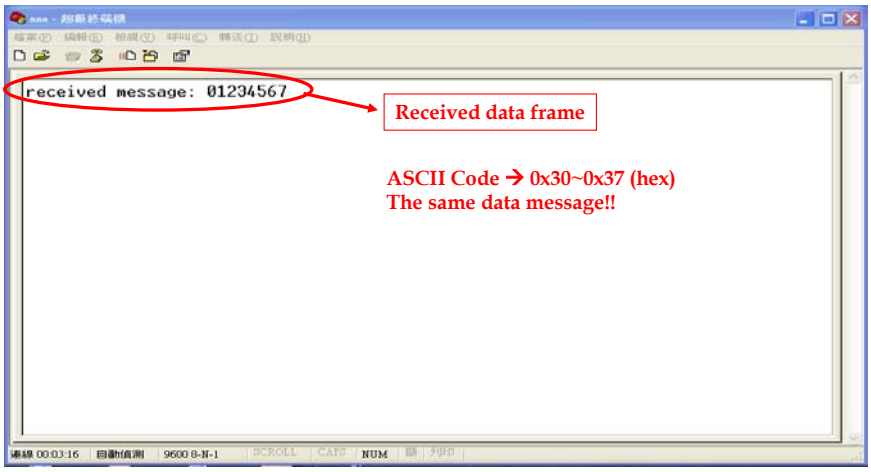


Figure 17 Experiment result of Figure 7-(a)



(a) Stage 1: data frame sent out by PC terminal (application layer)



(b)Stage 3: Data frame received by UART device

Figure 18 Experiment result of Figure 7-(b)

Another case on receiving for Figure 16-(b), there are three stages to be addressed. In first stage, the receiving case introduces that the data are transmitted from an application layer to CAN bus analyzer. Experiment result for transmitting a data frame contained with 8-bytes of length for size and ASCII code format is illustrated in Figure 18-(a). Through the ECAN-bus, the data frame is encapsulated firstly, and sent out from CAN-bus analyzer to Explore 16 development board in Stage 2. The data message can be easily received through EACN-bus network connection by ERIKA programming on the receiving case. Finally, the content of the data frame is returned and displayed on PC terminal (application layer) through UART device as shown in Figure 9-(b) in third stage.

Experiment results for Figure 17 and Figure 18 demonstrate that data messages can be sent and received successfully through ECAN-Bus network connection, which is mainly by programming the *TASKs* scheduling of *OSEK RTOS* of ERIKA. Therefore, ERIKA Enterprise provides reliable, easy designed and user-interface friendly programming environment in automotive industrial usage. The implemented characteristics by using ERIKA Enterprise are summarized in next section.

Although this paper is to make CAN bus work for real-time requirement, the experimental results cannot support this point. In our paper, we mainly adopt standard CAR-based OSEK operating system and porting it into the embedded system by open source from ERIKA. (e.g. A satisfiable case for automotive real-time embedded software system called *ERIKA Enterprise[11]* is a open-source RTOS implementation of the ISO 17356 API (derived from the OSEK/VDX API).) This is the focus of our paper. Actually, it is also difficult to have a complete demonstration for this work because of the most of things preserved by *OSEK real-time operating system (RTOS)*. If it needs to be proofed well for this work, it may be presented by another story which is not the scope of our paper. Thus, we merely consider an easily experiment instead of capacity of a printed page for our paper.

D. Implemented Characteristics

The implementation of ECAN-bus network connection in automotive industrial usage is important because of the transmission for data frame and signal. By following the standard of OSEK/VDX real-time operating system, ERIKA Enterprise is one better choice than other automotive softwares. The implemented characteristics of ERIKA Enterprise in automotive technology are mentioned as follows. (1) ERIKA Enterprise provides a tiny minimal system useful for automotive applications, as well as small embedded applications requiring real-time support. (2) Erika Enterprise implements various conformance classes, including the standard OSEK/VDX conformance classes *BCC1*, *BCC2*, *ECC1*, *ECC2*. (3) Erika Enterprise supports multicore partitioning of tasks into multicores. (4) ERIKA Enterprise also supports *Automatic* code generation. (5) ERIKA Enterprise aims to support heterogeneous multicore devices for the Automotive markets. (6) Industrial usages have been researched and applied in automotive technology. For example, these companies for *Cobra*, *Magneti Marelli* (e.g. Powertrain), *EnSilica*, *Aprilia Racing* (e.g. superbike engine controllers) are all supported by ERIKA Enterprise. (7) Various embedded evaluation boards are supported. (8) ERIKA Enterprise let the *user interface (UI)* be friendly.

Please note that something must be clarified as follows. (1) Our research work is mainly focused on an application of the motor control for ERIKA Enterprise via the CAN bus communication. Even though real-time CAN is a hot research area, it has not the difference from viewpoint of protocol. However, the traffic for CAN bus communication is attended by the characteristics of ERIKA's real-time. Therefore, the main story is for the framework research of the ERIKA Enterprise in this paper. Finally, a reliable message transmitting of CAN-Bus network connection applied in real-time CAR-based motor control system can be achieved by ERIKA's characteristics to achieve this goal. (2) The famous term for ECAN is a *Microchip dsPIC33F* trade mark. The ECAN module implements the CAN Protocol 2.0B, used primarily in industrial and automotive applications. Terms for CAN and ECAN for protocol viewpoint are the same. Actually, anyone developing drivers for CAN or

designing a network would require greater knowledge. Moreover, CAN was designed to work in a noisy environment such as an automobile or manufacturing facility. As such, extensive information in the serial data stream ensures that the data received are the data sent as already mentioned in the experiment of our paper from Figure 17 and Figure 18. Although this is a simple practice case, it is meaningful experiment for this paper. The advantage can be introduced by framework of ERIKA Enterprise as addressed in Section 4 which also adopts OSEK/VDX real time communication specification.

VI. CONCLUSIONS

In this paper, we present a reliable message transmitting for CAN-Bus network connection applied in real-time CAR-based motor control system. The research of the paper makes application users easily to understand in two ways: (1) traditional implementation for motor control (e.g. without ERIKA system), and (2) automotive software framework, programming design method, compilation and porting processes of the ERIKA. Moreover, we also propose a demonstrative application of the ECAN-bus network connection in ERIKA's programming designed method. The experiment results show that the data frames can have a real-time transmission and the ECAN-bus network connection is guaranteed by ERIKA Enterprise. For benefit of ERIKA, we merely focus on single core but not multiple cores in this paper, because it is easily for the migration from a single core to multiple cores, which means no changes to the application source code, only simple modifications to different OIL configurations. Please note that although there are also other embedded boards which can use open source software to send and receive CAN frames, this paper is mainly presented on a real time OSEK-based scheduling task. Therefore, through the research of this paper, the OSEK RTOS can be ported into target ECU hardware in very easy method. Also, the motor control can achieve the goal of the real-time management and reliability.

ACKNOWLEDGEMENT

This paper is an ITRI project, "Intelligent Green/Car Electronics Project", supported by Ministry of Economic Affairs, Taiwan, R.O.C. in granting number 102-EC-17-A-02-01-0691. Under this project, sub-projects are also available in granting numbers C367EM1100 and C352GC2310.

REFERENCES

[1] CAN Specification version 2.0. Robort Bosch GmbH, Postfach 30 02 40, D-70442 Stuttgart, 1991.

[2] S. Furst, "Challenges in the Design of Automotive Software," IEEE International Conference on Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 256-258, 2010.

[3] M. Broy, "Challenges in Automotive Software Engineering," the 28th International Conference on Software Engineering (ICSE), pp. 33-42, 2006.

[4] K. Grimm, "Software Technology in an Automotive Company-Major Challenges," IEEE the 25th International Conference on Software Engineering (ICSE), pp. 498-503, 2003.

[5] Official Web Page of the OSEK Project, http://www.osek-vdx.org/.

[6] Joseph Lemieux, Programming in the OSEK/VDX Environment, CMP Books, ISBN: 1-57820-081-4, pp. 1-359, Oct. 2001.

[7] Y. W. Li, H. W. Zhang, J. F. Gong and H. Rong, "Design of Automotive CAN Network Management Based on OSEK Standard," IEEE International Conference on Electronic and Mechanical Engineering and Information Technology (EMEIT), Vol. 2, pp. 717-721, 2011.

[8] C. W. Son, J. H. Kim, T. Y. Moon, K. H. Kwon, S. H. Hwang, and J. W. Jeon, "Analysis of Implementing OSEK NM with Two Types of ECU Network," IEEE International Conference on Control, Automation and Systems (ICCAS), pp. 575-580, 2008.

[9] X. Qiao, Z. X. Wang, Y. Sun, F. He and F. Y. Wang, "A CAN and OSEK NM Based Siren for Automobiles," IEEE International Conference on Networking, Sensing and Control (ICNSC), pp. 868-873, 2007.

[10] C. Nastasi et al., "Model based Real-Time Networked Applications for Wireless Sensor Networks," IEEE International Conference on Pervasive Computing and Communications (PerCom), pp. 1-3, 2009.

[11] Erika Enterprise web site, http://erika.tuxfamily.org/.

[12] R. Obermaisser, "Reuse of CAN-Based Legacy Applications in Time-Triggered Architectures," IEEE Transactions on Industrial Informatics, Vol. 2, Issue 2, pp. 255-268, 2006.

[13] Microchip for dsPIC33F Family Reference Manual Sections in, "Section 21. Enhanced Controller Area Network (ECAN™)", http://www.microchip.com, pp. 1-76, 2007.