An embedded communication platform based on Linux for automotive systems

Fernando H. Ataide[1], **Alan C. Assis**[1], **Carlos E. Pereira**[1], **Fabiano C. Carvalho**[2]

[1]Department of Electrical Engineering – Federal University of Rio Grande do Sul (UFRGS)
90035-190 Porto Alegre, RS, Brazil

[2]Department of Computer Science – Federal University of Rio Grande do Sul (UFRGS)
91501-970 Porto Alegre, RS, Brazil.

{fhataide,acassis,cpereira}@ece.ufrgs.br, fccarvalho@inf.ufrgs.br

Abstract. *In the last years, important research has presented different approaches in the real-time embedded communication domain aiming to cover the growing demands of performance, predictability and reliability of emerging applications. Such requirements involve low latency, reduced jitter, time composability, fault-tolerance and finally, support for future modifications. Mainly in the automotive domain which is considering the possibility of replacing the major part of mechanical and/or hydraulic systems for electronic systems, the importance of predictable behavior with some degree of flexibility plays a key role. This paper presents a prototype implementation of the FTT-CAN protocol over Freescale ColdFire platforms running RTAI as an embedded Linux – a work in the scope of the Baja-by-Wire[1] project which is being developed at Electrical Engineering Department of UFRGS. Some design issues together with latency and jitter results are provided and discussed. The project was developed entirely with open source tools.*

1. Introduction

Real-Time Embedded Communication Systems (RTECS) are becoming widely used in several areas of application, including automotive embedded systems. In this domain, severe timing constraints predominates hence the need of predictable service behavior in both time and value dimensions even in the presence of faults as the system can be strictly involved with either integrity of people or expensive equipment. For instance, nowadays there is a strong interest in the automotive industry towards the production of popular vehicles with electronic driving control, where conventional mechanic and hydraulic systems are replaced by electrical actuators and sensors focusing on weight diminution, cost and power consumption savings.
The main requirements involved in safety-critical applications are fixed protocol latency (low jitter), composability, support for fault-tolerance techniques and also some degree of flexibility in order to account for future modifications during the design phase. Considering that a distributed system is formed by a set of processing units which are spatially distributed in space, there must be a communication protocol providing reliable and timely services to the application layer. In this context, some existing protocols claim being able to fulfil these rigorous safety requirements. The first one is the TTP/C [TTA-Group 2003]

[1]This project is currently sponsored by FreeScale Semiconductor of Brazil.

protocol and its Time-Triggered Architecture (TTA) – a result of many years of work started in 1979 at the Technical University of Berlin with the MARS project. The other one is the FlexRay Communication System [Consortium 2004] developed by the FlexRay Consortium, formed by a group of strong companies in the automotive area.
A common characteristic of these protocols is their static communication behavior. Bus arbitration is essentially TDMA-based so the transmission times of all messages must be known in advance. To accomplish each network station has a fixed time slot for transmission that must be specified at design phase. In both TTP/C and FlexRay The definition of the slot time boundaries is fixed and cannot change during runtime. The FlexRay protocol in particular combines both time-and event-triggered traffic by defining a dynamic bus access window in which bus arbitration is performed in a FTDMA (Flexible Time Division Multiple Access) scheme thus providing some degree of flexibility. This feature is important for many of modern digital control systems (e.g. industrial control process and automotive systems) mainly when the controlled object dynamics cannot be completely defined before runtime thus requiring the control system to be adaptive. A RTECS with a high flexibility degree can provide a more efficient resource utilization, enhanced maintainability and easy integration/allocation of new functions in response to new functionality demands. The reader is referred to [Almeida et al. 2002] for a detailed overview about the main advantages of providing enhanced flexibility support in distributed control systems.
The FTT-CAN [Almeida et al. 2002] (Flexible Time-Trigerred Comunication on CAN) consists in a complete communication platform which was proposed as a solution to meet the flexibility requirement with guaranteed timeliness and efficiency. It combines a dynamic time-triggered together event-triggered traffic by means of temporal isolation of bus access time. In contrast to TTP/C and FlexRay, which have a static time-triggered schedule, all communication load can be dynamically scheduled at runtime. Unfortunately, the FTT-CAN is not commercially available yet. Nevertheless, its documentation is available in deep detail for system designers to implement the protocol.
Another important factor in this context is related to Real-Time Operating System (RTOS). In this project we adopt one to reach more reduction code size, maintenance improvement, reuse, and reduction of the implementation complexity. The choice of one suitable may be decisive in all life cycle of project.
This paper presents some practical results from a distributed system prototype in which the FTT-CAN protocol was implemented over RTAI as an embedded Linux running in a Freescale Coldfire embedded platform. The work is part of a drive-by-wire project, called Baja-by-Wire[1], which is being developed at the Electrical Engineering Department of UFRGS.
This paper is organized like follows. It starts with a brief presentation of the Baja-by-Wire project, followed by an overview of the FTT-CAN protocol. The proposed communication architecture with its mains components and functionalities is presented next. Finally, we discuss some practical results obtained from our prototypes followed by some conclusions and directions of future research.

2. The Baja-by-Wire Project

The Baja-by-Wire is a multi-disciplinary project which is being carried out by the Electrical Engineering Departments in cooperation with the Mechanics Department of UFRGS (Federal University of Rio Grande do Sul). The main goal is to develop a drive-by-wire dynamic system for an off-road race car as a digital distributed system with master and station nodes interconnected through a single serial bus. The FTT-CAN protocol was chosen as the underlying protocol due to its enhanced support for operational flexibility requirements. A simplified scheme of the car electronic system with its Electronic Control Units (ECUs) is depicted in Figure 1. The project lifecycle is divided in two phases. In

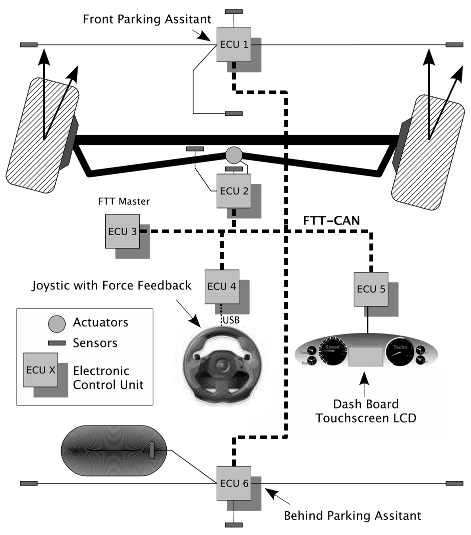

Figure 1. Steer-by-Wire System architecture

the first phase a functional prototype of a steer-by-wire system together with a dash board (instrumentation panel) and a parking assistant system may be developed. The purpose of the dash board is to collect telemetry and system diagnosis information such as speed, temperature, tacho and fuel level. Additionally, an interactive visualization of the parking assistant behavior is being considered. In the second phase the necessary fault-tolerance mechanisms must be included in order to ensure an appropriate dependability degree.

3. FTT-CAN Protocol

The FTT-CAN was proposed to cover the flexibility requirement on critical systems through online message admission control of time-triggered traffic without jeopardizing the overall system timing behavior [Almeida et al. 2002]. Admission control is performed by a central scheduler running at a particular node – the so called master node. It can receive dynamic transmission requests which are processed and evaluated by a feasibility algorithm. The time-triggered traffic is based on a relaxed master-slave medium access

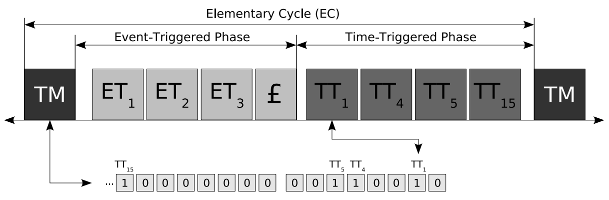

Figure 2. The Elementary Cycle (EC) in FTT-CAN.

control. A specific message – the Trigger Message (TM) – is transmitted by the master node in order to trigger the beginning of an Elementary Cycle (EC) inside which the

slave stations are transmit their messages either in the Event-Trigger (ET) phase or in the Time-Triggered (TT) phase. The TM carries schedule information that indicates which messages must be present in the next TT phase. Eventual bus access collisions between slaves messages are resolved by the native arbitration mechanism of the original CAN protocol.
The time-triggered message exchange of FTT-CAN follows a producer-consumer model whereas the event-triggered offers only send and receive basic services to be used when the application layer has aperiodic data to deliver. Slaves with pending aperiodic messages may transmit only during the ET phase which is the remaining time that is not used for time-triggered traffic in an EC. The transmission sequence is defined in the payload field of the TM message.
In FTTCAN, periodic message exchange takes place in an autonomous manner, i.e. the protocol stack proccess is responsible for the transmission of all messages inside the tt phase hence the application tasks do not need to invoke send and receive primitives.
In the ET-phase however messages are transmitted in response to explicit requests from the application layer.

4. The RTOS and Hardware Platform

The use of a RTOS is mandatory in certain circumstances when multiple instruction flows must be multiplexed for execution in a single processor. Depending on the complexity of the application, the use of a single execution flow to accommodate all system functions may result in unclear code and a time-consuming design phase. For that purpose a RTOS provides many advantages such as reduced code size, maintenance improvement, reuse, and so on.
Common features of a RTOS are: real-time scheduler, semaphores, interprocess communication mechanisms, interrupt handlers, among others. The absence of them means that the programmer himself would have to write low-level routines to handle hardware interfaces and devices. Thus, a RTOS provide means to implement more reliable code with reduced design turnaround time which is crucial for short time-to-market schedules.
For this project we decided to use an open source RTOS to run on the FreeScale ColdFire MCF5282 microprocessor platform. As a result of a comparative analysis the choice was to adopt the μClinux [Dionne and Durrant 2002] OS – a Linux operating system for microcontrollers without Memory Management Unit (MMU). An advantage of Linux-based operating systems are their modular kernel architecture which increases the flexibility of the system at runtime. In other words, kernel modules can be dynamically loaded for a particular situation, for instance, an Ethernet device driver and protocol stack can be loaded whenever necessary in order to provide gateway functionality over a short period of time.
Unfortunately, the μClinux by itself is not a RTOS. Its kernel is non-preemptive so there is no way to implement a real-time task scheduler. However, nowadays there are some projects aiming to provide real-time extensions for Linux, the most relevant of them being the RT-Linux [Yodaiken and Barabanov 2003] and the RTAI [Mantegazza 2001] extensions. The RTAI relies on the HAL (Hardware Abstraction Layer) concept and it considers the entire Linux kernel as a background task running when there is no real-time activity. It intercepts all hardware interrupts and routes them to either standard Linux or to real-time tasks. The reader is referred to [Andersson and Lindskov 2003] for an accurate comparative study between RT-Linux and RTAI.

For the purpose of this work the RTAI extension was considered the best solution. The factors that consolidated that choice were: 1) it is an open source project; 2) it has a mature code base; 3) there are free tools available, 4) small footprints; 5) active development community and 6) support to a wide variety of architectures such as x86, PowerPC, ARM, MIPS and m68k.
μClinux together RTAI gives us all present features of a RTOS that were presented above.

5. FTT-CAN over RTAI/μClinux Approach

Since the RTAI extension provide means to add time-critical functionality to the modular kernel of μClinux, the primitive functions of the FTT-CAN protocol were included in the Kernel Space Level (KSL) aiming to ensure proper real-time execution behavior. Likewise, on the User Space level (USL) it was decided to attach diagnostic tasks which are not time constraining. The logical architecture of the implementation, e.g. the modules stack, is depicted in Figure 3 in which the arrows demonstrate the relationships between its components. The RTAI extension has its own modules stack providing a set of basic services which allow the implementation of fully preemptive scheduling. However,

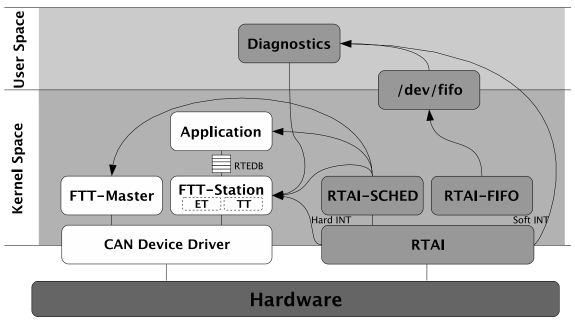

Figure 3. Modules stack.

for the purpose of this work only three modules were actually used. The first one is the `RTAI` module that implements the basic RTAI framework including interrupt dispatcher and low-level timer support. The second one is the `RTAI-SCHED` which provides preemptive scheduling of kernel modules tasks. Last, the `RTAI-FIFO` module consists in FIFO data structures and semaphores that are used for communication between the user and kernel spaces. On the left side the protocol stack is shown and on the top is the `Application` module that represents the process control tasks like sensor reading and calculation of output results. At the initialization procedure the kernel of slave nodes must be loaded with the `FTT-Station` module whereas the FTT-CAN master must be loaded with the `FTT-Master` module.
On the base of FTT-CAN stack there is a `CAN Device Driver` module which supply some methods to access an external, stand-alone CAN controller internal registers. It was decided to code a dedicated driver aiming to reduce additional overhead that would be generated if a COTS driver were used. The `CAN Device Driver` consists only in a restricted set of access primitives which provides means for both the `FTT-Station` and `FTT-Master` to interface with the controller.
The `FTT-Station` module is implemented as 2 high priority tasks (sync and async in Figure 3) and it is responsible for all message exchange on the system. For instance, it accesses data base either to update it with new data from the network or to load one sample

for transmission inside a CAN message. The RTAI interrupt service is the main resource of this module. Each time a new Elementary Cycle is started by the transmission of the TM an interrupt raises to trigger the execution of the `FTT-Station` module. Thereafter the contents of the TM is decoded for the module to know which RTE samples must be sent in the next time-triggered phase.

For a high level of abstraction the concept of Real-Time Entity (RTE) presented in [Kopetz 1997] was adopted. A RTE is any application variable whose validity is restricted in time. Each RTE data structure has the following attributes:

- Identifier (ID)
- Data size (DSize)
- Pointer to the data buffer (PDataBuff)
- A flag indicating whether it is produced or consumed (PC)
- Validity information over time (Tval)
- Pointer to its corresponding real-time task (Task)
- A flag indicating whether it is a time- or an event-triggered variable (EtTt)

The ensemble of RTEs of interest in a station node is allocated in the Real-Time Entity Database (RTEDB) representing the interface between application software and the protocol stack (Figure 3). The RTEDB is broken into two subtables aiming to optimize execution time since the `FTT-Station` module does not have to scan the entire RTEDB to search for outgoing data. Event-triggered messages are treated differently – they are queued in both sender and receiver sides. The queuing process is intelligent since it reorders the output sequence buffer according to the message priority defined by its identifier field. This prevents high priority messages from being blocked in the queue due to low priority ones that lost successive transmission attempts. The creation of the RTEDB in memory is performed during the startup phase. As the set of RTEs in the application process is modified any required changes in the RTEDB can take place at runtime.

The temporal isolation between the TT and ET phases is guaranteed by means of a global offset from the transmission of the TM. Once the `FTT-Station` is informed about the reception of the TM it starts an internal timer in order to schedule the beginning of the following time-triggered phase at the appropriate point in time. ET messages which have been previously load for transmission are sent if and only if there is enough time before the tt phase begins. This is implemented by enabling/disable the corresponding output buffer of the CAN controller.

On Message exchange level of time-triggered phase the control is autonomous, i.e. the transmission and reception is performed by FTT-CAN protocol level without interference from of application level. The transmission sequence is carried out through a buffer defined at TM message decoding moment. In event-triggered phase the message exchange has implict requests of application level through a specific API in the external control way. Each request is queueing in agreement message identifier which define the message priority. There are two distinct queue to event-triggered phase, one to RTE event-triggerd process variable and other to simple messages.

Finally, the `FTT-Master` module is responsible for the dynamic message scheduling of the TT phase. Prior to sending the TM message it defines the sequence of messages to be transmitted based on a schedulability analsys method that takes into account timing requirements like period T_i, priority P_i, worst-case transmission times C_i, relative phase Ph_i and deadline D_i. On the other hand aperiodic traffic is scheduled based on fixed priority methods.

6. Practical Results

The RTAI interrupt latency and jitter has direct impact on the protocol response times. However, in this current implementation the resulting interrupt latency is near 21μs with an associate jitter of 4μs which is relative low when comparing to the timing requirements of the application.
For a jitter evaluation of the FTT-CAN communication platform we consider a set of messages as different periods showed in the Table 1. This set was based on baja-by-wire application (Figure 1). Each message was sampled by 3000 successive occurrences. The

Table 1. Set of messages for evaluation

	Name	Description	Node	P_i	T_i	Ph_i
1	SteerWheelAngle	Angle for actuation on wheels	4	7	5	0
2	WheelAngle	Current Angle of the wheels	2	6	5	0
3	WheelSpeed	Current speed	2	5	5	0
4	EngTemperature	Current engine temperature	1	4	100	50
5	FuelLevel	Current level fuel	6	3	100	50
6	FrontParking	3 bits value of front sensor	1	2	200	100
7	BehindParking	3 bits value of behind sensor	6	1	200	100

EC has the minor period of the set message, in this case 5ms. The Figure 4 (a) showed the bus signal of a EC captured by an osciloscope. The Time-Triggered phase was placed in the end of EC that is programmed to start in accordance as TM content, in this case of specific sample it was programmed to hold five Time-Triggered message. For the better bandwidth use the the message 6 and 7 are added on Time-Triggered phase only in parking situations through admission control process. The Figure 4 (b) showed the measured

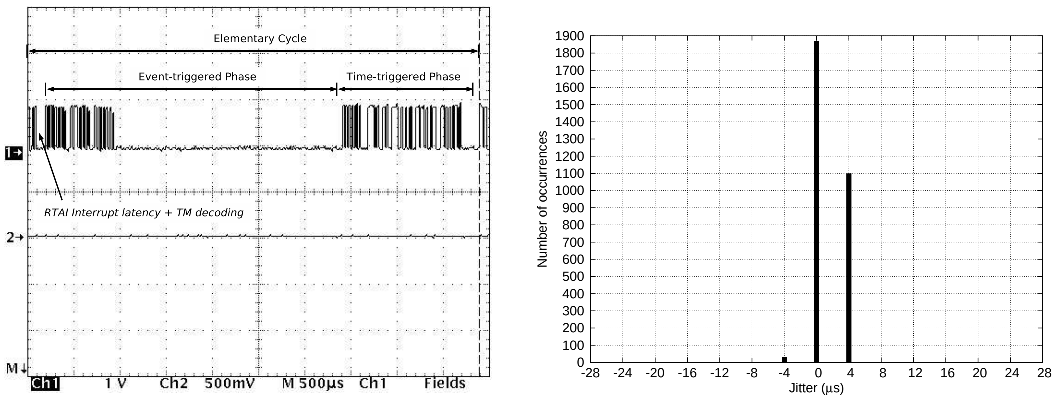

Figure 4. (a) Bus signal image of a Elementary cycle and (b) jitter distribution of the TM message.

jitter of TM. It is inherent of the RTAI scheduler that active periodically the master dispatcher task, thus being very difficult to improve it. Although it is a excellent result for this intention.
The TM jitter and its interrupt handler jitter has a direct impact on Time-Triggered messages because the TM decoding that is intrinsically in them. The Figure 5 (a) and (b) showed the jitter distribution of messages 4 and 7, respectively.
The Figure 6 (a) and (b) showed the jitter distribution of messages 1 and 3, respectively.

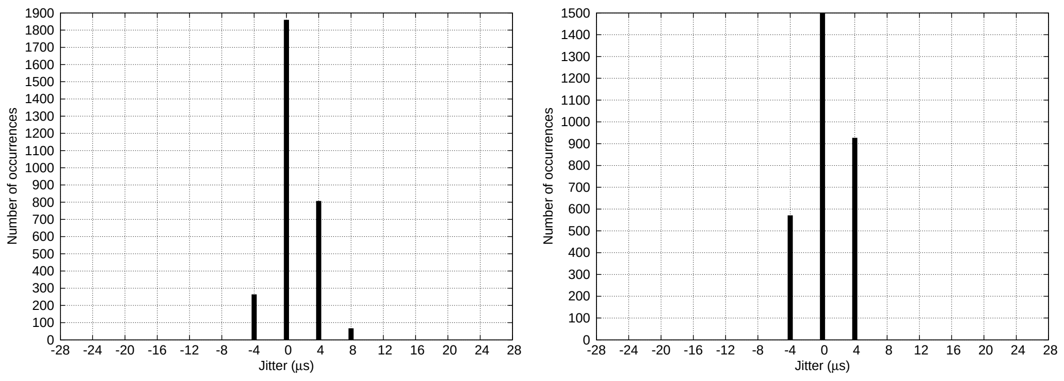

Figure 5. (a) Jitter distribution of a synchronous message as 100ms and (b) as 200ms of period .

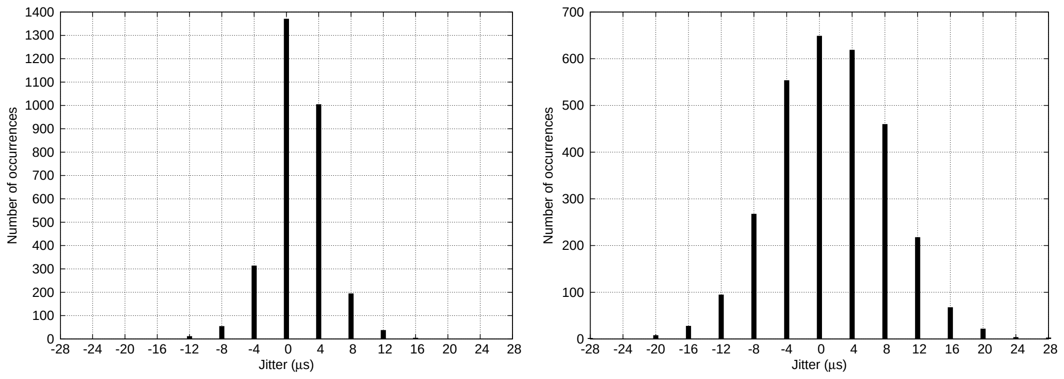

Figure 6. Jitter distribution of the first (a) and last (b) synchronous message as 5ms of period.

The bit stuffing mechanism of CAN protocol contributes for this small jitter increase. This measured jitters is insufficient for to cause a performance degradation on the control system presented in this paper.

7. Conclusions and Future Work

The FTT-CAN implementation may be directly affected by RTOS in use. However, is very important the choice of a suitable RTOS with hard precision on the scheduler and interrupt handler. The RTAI/μClinux has presented excellent performance for hard Real-Time application. Analyses of a method for admission control of messages on the system in run-time phase and analyses of differents approaches about holistic scheduling are some of future works.

The use of open source solution provides some benefits which would not have using proprietary solutions. The support is found on dedicated discussion lists and its members may to contribute actively in the evolution of project offering improvements in its source code. And this way contributing in the intellectual growth of everyone. This is open source.

8. Acknowledgments

This work has been partly supported by the Brazilian research agency CNPq and FreeScale Semiconductor of Brazil.

References

Almeida, L., Pedreiras, P., and Fonseca, J. (2002). The ftt-can protocol: Why and how. *IEEE Transactions on Industrial Electronics*, 49(6):1189– 1201.

Andersson, M. P. and Lindskov, J.-H. (2003). *Real-Time Linux in an Embedded Environment - A Port and Evaluation of RTAI on the CRIS Architecture*. Master of Science Thesis, Lund Institute of Technology, Sweden.

Consortium, F. (Copyright 2004). *FlexRay Communications System Protocol Specification Version 2.0*. FlexRay Consortium.

Dionne, J. and Durrant, M. (2002). Embedded linux/microcontroller project.

Kopetz, H. (1997). *Real-Time Systems - Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers.

Mantegazza, P. (2001). The diapm rtai project homepage.

TTA-Group (2003). *Time-Triggered Protocol TTP/C High-Level Specification Document Protocol Version 1.1*. TTA-Group.

Yodaiken, V. and Barabanov, M. (2003). A real-time linux.