Using FTT-CAN to the flexible control of bus redundancy and bandwidth usage[1]

Valter Silva, ESTGA/DETI/IEETA - Universidade de Aveiro

José Alberto Fonseca, DETI/IEETA - Universidade de Aveiro

Joaquim Ferreira, DETI - IP Castelo Branco

Controller Area Network (CAN) is a popular and very well-known bus system, both in academia and in industry, initially targeted to automotive applications as a single digital bus to replace the wiring that were growing complexity, weight and cost with the advent of new automotive appliances. However, requirements have evolved and CAN's dependability and bandwidth limitations led to the emergence of alternative networks such as FlexRay and TTP/C. Nevertheless, we believe that it is possible to improve CAN so it could fulfill contemporary requirements. This paper proposes the use of Flexible Time-Triggered CAN (FTT-CAN) to increase the available bandwidth while providing fault tolerance in CAN based systems with multiple buses. The architecture and flexibility of FTT based systems enables a tight yet flexible control of redundancy and bandwidth usage without increasing the complexity of the nodes. In this novel solution, a FTT-CAN Master controls the dispatching of messages among a set of independent buses. The Master can react online to bus failures switching the transmission of critical messages to a non-faulty bus, always keeping a pre-determined redundancy level.

1 Introdution

Distributed embedded systems (DES) have been widely used in the last few decades in several application fields, ranging from industrial machinery to avionics, automotive systems and robotics. Most of these applications require precise and fault-tolerant temporal coordination among the nodes of the DES. Time-triggered communication protocols are usually adopted to fulfill this requirement and most of these protocols also have mechanisms to provide fault-tolerance and online reconfiguration. However, the inclusion of these features also increases the computing and network overhead thus contributing to reduce the traffic payload and increasing the complexity of the nodes or the protocol.

FTT-CAN [1] is a synchronous time triggered protocol built on top of CAN. It allows the coexistence of synchronous and asynchronous traffic with fault-tolerance and reconfiguration mechanisms using just one CAN fieldbus. However, it introduces some network overhead due to the use of control messages. The computational overhead introduced by the middleware also limits the maximum bit rate of the CAN fieldbus when nodes based in low performance microcontrollers are used. Thus, the CAN bandwidth is not fully exploited in FTT-CAN, unless high performance processors are used, typically above 16 bit, 40 MHz.

The overhead penalty introduced by FTT protocol, together with CAN 1 Mbps bandwidth limitation makes, FTT-CAN unsuitable for applications that require higher bandwidth, e.g., video traffic. Currently, the maximum bit rate attainable is 250 kbps with an implementation using PIC microcontrollers [2].

FTT-CAN is currently used in mobile robotics applications [2]. In this application field, the use of sensors demanding high bandwidth and nodes with low processing

[1] This work was supported by *Fundação para a Ciência e Tecnologia* under grant PRODEP 2001 – Formação Avançada de Docentes do Ensino Superior Nº 200.019 and by ARTIST2, NoE on Embedded Systems Design, (EC-IST - IST-004527).

power controllers is common. A preliminary study of the bandwidth usage of sensors in autonomous mobile robots is presented in [3]. Some examples of robots with sensors demanding high bandwidth can be found there. In [4], a sonar sensor, used for navigation, uses almost all the bandwidth of the CAN fieldbus. Other frequently used sensors in robotics are video cameras which also have high bandwidth requirements.

Systems like TTP/C [5] and FlexRay [6] provide dependability for safety critical applications by using a redundant communications medium where redundant messages are transmitted. In these systems the redundant bus can also be used to increase the available bandwidth by enabling the transmission of non-redundant traffic in the redundant buses. However, these systems have just one redundant bus with static offline message scheduling.

Recent work by Pimentel [7] proposes redundant CAN-based systems with redundant buses, interfaces, ECUs (Electronic Control Units) and message transmission. A simplified middleware interfaces with the application, taking care of the redundant message transmission and of selecting one of the redundant messages received. An example of a steering by wire system [8] is used for demonstration purposes. This solution, however, does not increase the total available bandwidth. In this paper we propose to take advantage of FTT-CAN flexibility and synchronization properties, to combine fault-tolerance with bandwidth increase. This is done by using N redundant buses in which the traffic is scheduled on-line by a FTT-CAN Master. Depending on the application, it is possible and easy to implement features that contribute to manage fault-tolerance and bandwidth usage such as to control redundant transmission of critical messages, to react to bus failures keeping a K-level redundancy, to use the available buses in the absence of failures to transmit non-critical traffic, to switch on-line to degraded operation modes in the presence of bus failures. This is done with a limited increase in the complexity of the FTT-CAN Master and a very slight increase in the complexity of the Stations (also know as slaves). It should be noticed that, although this proposal increases the responsibility of FTT-Masters in what concerns fault-tolerance management, they have already been object of extensive work concerning their own dependability [9]. Thus, the proposed solution relies in components which are rather robust from the dependability point of view.

This paper is a follow-up of the work presented in [15] and presents a solution to implementation the master node using a desktop computer with a PCI CAN board.

The rest of the paper is organized as follows:

In the next section the FTT-CAN paradigm is briefly presented and discussed. In section 3 the problem of FTT-CAN bandwidth limitation is presented and in section 4 a solution for this problem is presented. In section 5, techniques for dispatching trigger messages are discussed based on previous approaches. In section 6 the master implementation using a PC with a PCI-CAN board is discussed. Finally, the paper ends with conclusion section.

2 FTT-CAN brief presentation

The FTT-CAN protocol (Flexible Time-Triggered communication on CAN) [1] has been developed with the main purpose of combining a high level of operational flexibility with timeliness guarantees. It uses the dual-phase elementary cycle concept for isolated time and event-triggered communication. The time-triggered traffic is scheduled on-line and centrally in a particular node called a master, facilitating on-line admission control of requests, thus being managed in a flexible way, under guaranteed timeliness. The protocol relies on a relaxed master-slave medium access control in which the same master message triggers the transmission of messages in several slaves simultaneously (master/multi-slave). The eventual collisions between slaves' messages are handled by the native distributed arbitration of CAN.

The next figure depicts the general architecture of the system. Note that more

than one master and more than one station can be used to provide redundancy.

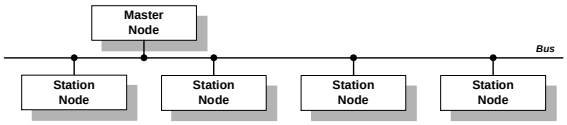

Figure 1: General architecture

FTT-CAN slots the bus time in consecutive Elementary Cycles (ECs) with fixed duration. All nodes are synchronized at the start of each EC by the reception of a particular message known as an EC trigger message (TM), which is sent by the master node. Within each EC the protocol defines two consecutive windows, asynchronous and synchronous, that correspond to two separate phases (see figure 2). The first is used to convey event-triggered traffic, here called asynchronous because the transmission requests can be issued at any instant. The second is used to convey time-triggered traffic, herein called synchronous because its transmission occurs synchronously with the ECs. The synchronous window of the nth EC has a duration that is set according to the traffic scheduled for it. The schedule for each EC is conveyed by the respective EC trigger message (see figure 3). Since this window is placed at the end of the EC, its starting instant is variable and it is also encoded in the respective EC trigger message.

The communication requirements are held in a database located in the master node [10], the System Requirements Database (SRDB). This database holds several components, one of which is the Synchronous Requirements Table (SRT), that contains the description of the periodic message streams. Based on the SRT, an on-line scheduler builds the synchronous schedules for each EC (EC schedules). These schedules are then inserted in the data area of the appropriate EC trigger message (see Figure 3) and broadcasted with it. Due to the on-line nature of the scheduling function, changes performed in the SRT at run time will be reflected in the bus traffic within a bounded delay, resulting in flexible behaviour.

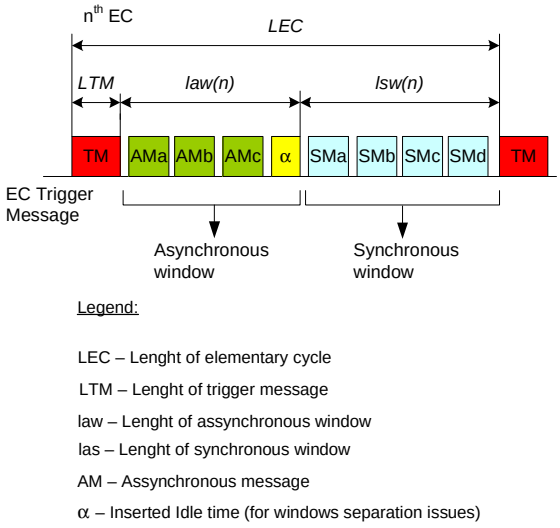

Figure 2: The elementary cycle in FTT-CAN

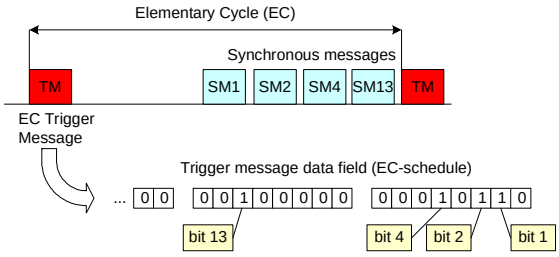

Figure 3: Master/multislave access control and EC schedule coding scheme

It should also be noticed that, in FTT-based systems, the slave nodes are not allowed to retransmit synchronous massages affected by errors, to prevent cascading transmission errors [11].

3 The bandwidth problem

In CAN 2.0A, the frame has 11 bits for message identification, 57 bits for control and stuff bits.

A message with the maximum length (8 Byte) can have 132 bits [12] including all the stuff bits, control and identification. Thus, with a usable data of 64 bits, the message occupies 132 bits, leading to an overhead of more than 50%. If the CAN message is 1 byte long, the payload is just 8 bits in a message which can be 62 bits in total, leading to an overhead of 87%. This means that, if a 1Mbps rate is used in the fieldbus, in the better case, just 500 kbps are available for payload data. This

analysis is even worse when using FTT-CAN.

The FTT-CAN protocol uses a CAN network which can run up to 1 Mbps. Therefore, like said previously, some of the available bandwidth is dedicated to the CAN protocol overhead. The FTT-CAN protocol also increases this overhead because it needs a synchronization message every Elementary Cycle. The overhead of the trigger message is 2.6% for a bit rate of 1Mbps and a Trigger message with 8 data bytes, and an EC of 5 ms. For the same parameters, but using 125 kbps, the overhead of the trigger message is 21.1%. For 250 kbps the trigger message overhead is about 10%. This bit rate is the limit in the reference implementation in PIC microcontrollers running at maximum speed.

The available bandwidth in the CAN fieldbus is also limited by the bus length. The maximum length for a bus running at 1 Mbps is 40 m. This limitation can be important if the bus will be used in an industrial environment or in an aircraft.

The CAN bus is not error free. In [13] bit errors assessment is made.

On the other hand, fault tolerance techniques developed for FTT-CAN do not contemplate bus failures, they just solve the problem of the failure of the master(s) and of protecting the bus from incorrect transmission made by the slaves [11], [14]. Thus, if the bus fails, all the system fails.

4 Proposed architecture for redundancy and bandwidth improvement

To overcome the limitations referred in the last section, a novel architecture capable of handling synchronous messages replication has been proposed by Silva *et al.* [15].

The system architecture is presented in next figure.

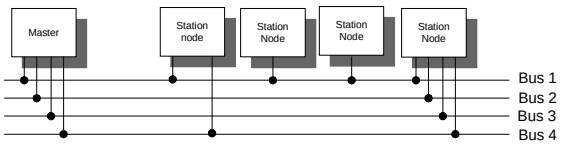

Figure 4: System architecture

In figure 1 the general architecture of the first release of FTT-CAN is presented. In this novel architecture more than one bus is used to permit redundant messages sending the same message in several buses. Moreover, the buses can be used to increase the available bandwidth available in the system sending different messages in different buses.

Note that the master node connects to all the buses in order to control all of them. However, due to communications requirements or even due to economical reasons, the slave nodes can connect to just one bus or a set of buses. In case salve nodes are connect to just one bus, they are unable to take advantage of improved bandwidth (to send redundant messages or to send different messages). However, they are simple and can be equal to the nodes used in the first release of the FTT-CAN.

The master redundancy techniques presented before by Ferreira *et. al.* [16] is still valid in the proposed architecture. It is still possible to use slave redundancy and bus guardians. However in this paper no further considerations on this issue will be made.

The hardware architecture of the new system is presented in Figure 5.

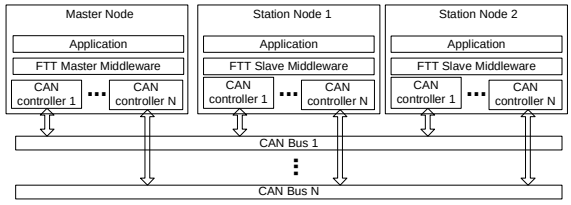

Figure 5: Hardware architecture

Each node (Master or Slave) has a layer (FTT layer) responsible for the management of the available CAN controllers while providing services to the application layer. These services in the master node are to manage the synchronous messages and in the slaves is the management of the synchronous and asynchronous messages.

As presented in Figure 3 the master node must issue Trigger Messages to the bus. These messages are for time slotting and contain the information about the synchronous messages that will be

transmitted in the current Elementary Cycle.

5 Trigger message dispatching

If just one CAN bus is used, the master schedules the messages and, after, dispatch the Trigger Message to the bus. If more than one bus is used the master must issue a trigger message per each bus it is connected to. In this case several approaches can be used for the dispatching of the trigger messages and the meaning of the trigger flags [15]. For this paper we assume the use of a different trigger message in each bus. This means that each bus can be viewed at a self contained bus improving the flexibility for scheduling made by the master.

For the trigger flags, also, several approaches can be used. To have an efficient use of the available trigger flags the strategy we will use further in this document is each trigger flag has an independent meaning in each bus. The next figure presents an example of this strategy using three buses.

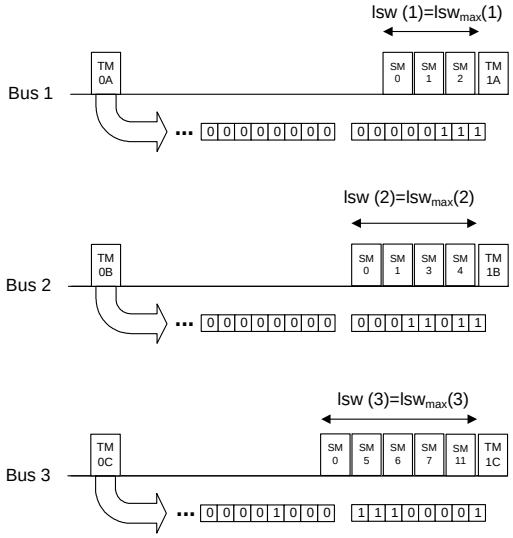

Figure 6: Trigger message format and dispatching

In the figure 6, trigger flag 0 is issued in the three buses. This means, the corresponding producer of message with identifier 0 in bus 1 must send the message in the current EC. However, other producer, or the same, must send a message with identifier 0 in the bus 2. This message can be the same message as in bus 1, or not. This means that each message is identified by its CAN identification and the bus where it is send.

Thus, the redundancy control and replica management are performed by the application layer (see Figure XXX) and is not responsibility of the FTT layer. The application is responsible for the correct use of the available buses and CAN identifiers. Moreover, the application must have a-priori knowledge of the buses the nodes are connected to (please recall that the slaves can connect to just one bus or a set of buses). However, they must connect to all available buses in order to control the total bandwidth in the system.

6 Master implementation

The available slaves for the first release of FTT were developed in a Microchip PIC18F258. This microcontroller provides 32Kb of Flash memory, 1.5Kb of RAM memory and one CAN controller. The master was also implemented in this microcontroller. However, because it just has one built-in CAN controller this microcontroller is not suitable for the master of the new FTT-CAN implementation. It is necessary a microcontroller with more than one built-in CAN controller, or an external CAN controller that can be connected to the microcontroller. Connecting an external controller requires more software and hardware, low latency access to the internal registers of the controller, and so, we will not considerer this option further.

The slaves can use just one bus thus, the same hardware platform was used in the scope of this work.

The master for the present version was implemented in a desktop computer with a PCI-CAN board from EMS [17]. This board connects to the PCI bus of the computer and has two Philips SJA1000 CAN controllers.

The Desktop computer runs the RTAI with 2.6.9 kernel and Linux operation system (Fedora Core 3 distribution). RTAI is a freeware application interface with real time support for Linux.

The global architecture of the software is presented in Figure 7.

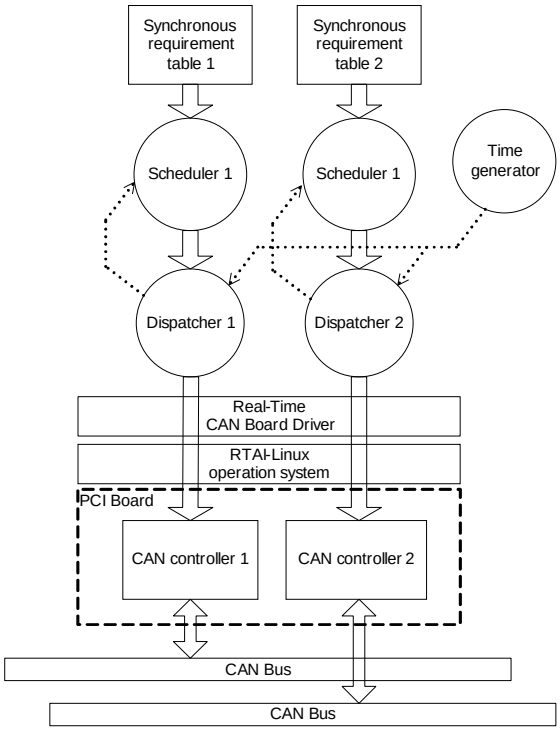

Figure 7: General master architecture

The master node has a Synchronous Requirement Table, a Scheduler and a Dispatcher per each available CAN controller. Both the dispatchers and the scheduler work independently from each other.

Based on the synchronous requirements, the schedulers build an independent schedule for the next EC. The dispatchers, using the time information provided by the timer generator, dispatch the different trigger messages in each bus the same time.

The RTAI operation system provides functionalities for task and time management, for accessing to the PCI bus and to manage the hardware interrupts. The schedulers and the dispatchers were developed in the kernel space to have minimum interference from the operation system and application interference.

The timer manager is a periodic task which triggers the dispatcher that, in turn, triggers the scheduler to build the schedule for the next Elementary Cycle.

The real-time driver for the PCI CAN board was developed in the kernel space using the facilities of the RTAI application interface to access to PCI bus and to manage the hardware interrupts.

Currently, we are assessing the performance of the master implementation in the PC, using the PCI-CAN board.

Some experiments to measure the time difference between two “parallel” trigger messages are being designed. The computing time of two “parallel” scheduling operations is also being assessed.

7 Conclusions

This paper presents a new architecture to improve the bandwidth and the fault-tolerance of FTT-CAN. To achieve this objective, several CAN buses were used, either transmitting the same message in different buses (spatial redundancy) or different messages in different buses, increasing the available bandwidth.

The slaves can be the same used in the early version of FTT-CAN, with just one CAN controller. Thus, the total available bandwidth is not used by each slave individually. Moreover, the slaves also can use more than one bus to transmit in more than one bus. However, if just one bus is used, the slaves remain simple (are the same of the early versions of FTT-CAN) and with low price.

On the other hand, the master node must control all the available buses. Thus, a processor or microcontroller with more than one CAN bus or an external controller is necessary to develop it. In this work a desktop computer with a PCI CAN board have been used. The implementation of the master firmware is done using the functionalities provided by RTAI to manage timer, the PCI bus and the hardware interrupts.

In the future, the implementation of the slave nodes will be done in a desktop computer. The slave node and the master will be implemented in a low cost microcontroller from microchip, the dsPIC 30F6012A, which has 2 built in CAN controllers.

References

[1] Almeida, L., Pedreiras, P., Fonseca, J.A.G., "The FTT-CAN protocol: Why and how", IEEE Transactions on Industrial Electronics, Volume 49, Issue 6, Dec. 2002, pp. 1189-1201.

[2] Silva, V., Marau, R., Almeida, L., Ferreira, J., Calha, M., Pedreiras, P., Fonseca, J., "Implementing a distributed sensing and actuation system: The CAMBADA robots case study", in Proc. ETFA 2005, September 2005, pp. 781-788.

[3] Silva, V., Fonseca, J., Nunes, U., Maia, R., "Communications Requirements for Autonomous Mobile Robots: Analysis and Examples", in Proc. FeT 2005, November 2005, pp. 91-98.

[4] Mendes, A., "Detecção e seguimento de alvos com Laser Range Finder", MsC Thesis, University of Coimbra, 2004.

[5] Kopetz, H., Grunsteidl, G., "TTP-A protocol for Fault-Tolerant Real-Time Systems", computer, vol. 27, issue 1, Jan 1994, pp 14-23.

[6] FlexRay, "FlexRay Communications System Protocol Specification", version 2.1, 2005.

[7] Pimentel, J., "Safety-Riability of Distributed Embedded Systems Fault Tolerant Units", in Proc. IECON 2003, November 2003, vol. 1, pp. 945-950.

[8] Pimentel, J., "An Architecture for a Safety-Critical Steer-by-Wire System", SAE congress 2004, paper nº 0714.

[9] Ferreira, J., Pedreiras, P., Almeida, L., Fonseca, J., Achieving fault tolerance in FTT-CAN", in Proc. WFCS 2002, August 2002, pp. 125-132.

[10] P. Pedreiras, "Supporting Flexible Real-Time Communications on Distributed Systems", PhD Thesis, University of Aveiro, Portugal, July, 2003.

[11] Ferreira, J., "Fault-Tolerance in Flexible Real-Time Communications Systems", PhD Thesis, Univerisity of Aveiro, 2005.

[12] Nolte, T.; Hansson, H.; Norstrom, C., "Probabilistic worst-case response-time analysis for the controller area network", In Proc. 9th IEEE Real-Time and Embedded Technology and Applications Symposium, May 2003, pp. 200-207.

[13] Ferreira, J., Oliveira, A., Fonseca, P., Fonseca, J., "An Experiment to Assess Bit Error Rate in CAN", in Proc. RTN 2004, pp. 15-18.

[14] Marau, R., Silva, V., Ferreira, J., Almeida, L., "Assessment of FTT-CAN master replication mechanisms for safety-critical applications" to appear in SAE congress 2006, paper nº 06AE-278.

[15] Silva, Valter F., Fonseca, José A., "Using FTT-CAN to combine redundancy with increased bandwidth", In proc. of 2006 IEEE International Workshop on Factory Communications Systems, pp. 55-63.

[16] Ferreira, J., Almeida, L., Fonseca, J. A., Pedreiras P., Martins, E., Rodríguez-Navas, Rigo, J., Proenza, J., "Combining Operational Flexibility and Dependability in FTT-CAN," IEEE Transactions on Industrial Informatics, vol. 22, no. 2, May 2006.

[17] EMS web site. available at: http://www.ems-wuensche.com/