Design and Implementation of a Redundancy Manager for Triple Redundant CAN Controllers

Carlos Guerrero, Guillermo Rodríguez-Navas and Julián Proenza
Dept. de Matemàtiques i Informàtica
Universitat de les Illes Balears
Edifici Anselm Turmeda, Campus UIB
07071 Palma de Mallorca, SPAIN
: *guerrero@ipc4.uib.es, vdmigrg0@uib.es, dmijpa0@ps.uib.es*

Abstract— **There is a growing interest in using the *Controller Area Network* (CAN) protocol for critical control applications. In many articles studying this possibility, it is assumed that CAN controllers, which are the circuits that implement most of the protocol specification, never present faults. In this paper we present the architecture of a fault-tolerant CAN controller subsystem that allows substantiation of this assumption. Our subsystem is made up of three standard CAN controllers and a specifically designed circuit, called *Redundancy Manager* (RM), which guarantees the coordinated operation of the introduced redundancy. Said circuit has been designed to adapt to the specific characteristics of the CAN protocol and is totally compatible with other fault tolerance mechanisms that have been designed in the past for other parts of the system. After presenting the behaviour and architecture for the fault-tolerant CAN controller subsystem, we present a complete design for the RM. This design has been implemented and simulated using a VHDL tool. Results of this simulation are also shown.**

I. Introduction

The *Controller Area Network* (CAN) protocol [1] is a serial bus with a maximum bit rate of 1Mbit/s. Originally developed by Bosch for in-car communications, CAN is nowadays used in a wide range of applications, including industrial automation, due to the dramatic reduction of price that its components have experienced in the last years. Among those components, the so-called CAN controllers are typically responsible for the implementation of most of the protocol specification.

Besides its reduced cost, CAN also presents other relevant merits, particularly in relation to real-time performance [2] and fault-tolerant operation. Due to these attributes there is a growing interest in using CAN for critical control applications [3], [4], [5], [6], [7], where dependability is a fundamental goal. Following this tendency, the time-triggered extension of CAN called TTCAN [8] has been developed as an alternative to TTP [9] in very critical automotive x-by-wire applications. These new applications of CAN require an improvement of its fault-tolerance capabilities.

In many articles on dependable CAN networks, it is assumed that CAN controllers never present faults [3], [4]. However this assumption is incorrect in relation to the well-known limited reliability of electronic components. In order to allow substantiation of such an important assumption we have designed a fault-tolerant CAN controller subsystem, which basics have been already introduced in a previous work [10]. This subsystem is made up of three standard CAN controllers and a specifically designed circuit, which is intended to manage the introduced redundancy. The resultant redundant controller works even in the presence of any permanent fault in one of the controllers, and can tolerate transient faults without redundancy attrition. Our approach achieves CAN controller fault-tolerance in such a way that it is then unnecessary to modify other parts of the system, different from the CAN controller subsystem. This makes our design totally compatible with other fault tolerance mechanisms which have been proposed for other parts of the system [3], [4]. In this paper both the behaviour and the architecture of our fault-tolerant CAN controller subsystem will be described.

The redundancy management circuit has been designed to adapt to the specific characteristics of the CAN protocol [1]. Some of these characteristics will now be explained as they are particularly relevant for the work we are reporting on. First, all the nodes, even the transmitter, sample each bit on the bus. In this sampling, all nodes will simultaneously see the same bit of the stream transmitted through the bus. Second, the two logical values that can be transmitted are called *dominant* and *recessive*. When one transmitter sends a dominant value, the bus will take this value independently of the other nodes' contributions. In contrast to this, a recessive value is only visible in the bus when all nodes are sending recessive values. Third, several error detection mechanisms allow all nodes to detect when a specific bit of the stream is erroneous. One of those mechanisms is the bit stuffing rule, which specifies that after a sequence of five consecutive bits with the same polarity a bit of the opposite polarity must be inserted. Fourth, in the event of an erroneous bit, all nodes notify the situation to the other nodes by transmitting a sequence of six consecutive dominant bits, called *error flag*, that will cause additional erroneous bits in nodes that have not detected the first erroneous bit. Then all nodes reject the frame affected by the errors in order to preserve the *data consistency*. Fifth, after this error signalling, the frame affected by the errors is scheduled for retransmission. Simultaneously, all nodes increment their error counters. Sixth, when error counters reach specific values, the corresponding node is disconnected from the bus for *error-containment* [11] purposes. Due to limitations of space we suggest readers may look up a more detailed description of all these mechanisms [1].

II. Proposed Redundancy Technique

The structure of a typical non-redundant (simplex) CAN node is shown in Fig. 1. As can be seen, the basic components are a processor, typically a microcontroller, which executes the application; a CAN controller, which implements most of the CAN protocol, and a transceiver, which simply adapts the electronic signals for their transmission and reception through the communication medium. Note that, at the implementation level, the processor and the CAN controller are sometimes included in a

0-7803-7474-6/02/$17.00 ©2002 IEEE

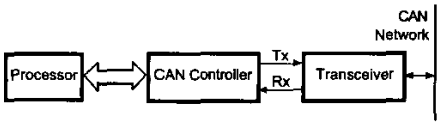

Fig. 1. Non-redundant CAN node architecture

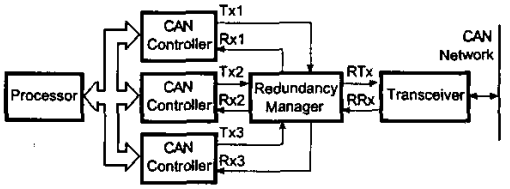

Fig. 2. CAN node with triplicated CAN controller

single integrated circuit.

In this paper, we focus on providing tolerance to faults in the CAN controller, particularly to those manifesting in the interface between the controller and the communication medium. As can be seen in Fig. 1 this interface is constituted by two signals, Tx and Rx, each one representing the logical values transmitted to the medium and received from it, respectively. Tx is used when the controller is in the transmitter state or when error signalling has to be performed. On the other hand, Rx is used to sample the bits in the medium both in the receiver and transmitter states, in the last case, for error detection purposes. The value sampled in Rx is the value of what we will call the *channel*, which constitutes the logical AND of all the contributions from the different nodes, provided that the recessive value is associated to the logical '1' and the dominant value to the logical '0'.

The fault model we are going to consider is quite simple. Any relevant fault of the CAN controller will manifest itself, sooner or later, as a value in the Tx output differing from what is expected. This can happen in one or more bits of a frame or of the interframe space. Some of these errors may violate the CAN protocol in ways that would be detectable by the CAN's corresponding mechanisms. However there are many CAN controller faults that would not lead to this kind of errors. For instance, a fault in a transmission buffer of the controller could change the value of one of the data bits which are to be transmitted. If the rest of the controller is non-faulty the frame transmitted in the end would conform with the CAN protocol rules and thus the error would be undetectable by the CAN's error detection mechanisms. On the other hand, faults manifesting as errors in the transmission of information from the CAN controller to the local processor are beyond the scope of this work.

Our approach to fault tolerance is based on triplicating the CAN controller so that any single permanent fault in one of the controllers will be tolerated. The proposed architecture for the redundant CAN controller is shown in Fig. 2. Note that, although a CAN node presents other components that could fail, this work is not intended to provide fault tolerance in such other components and, as a consequence, redundancy only appears at the controller level. In Fig. 2, the connection between the processor and the redundant controller is shown with a minimum of detail. In fact there is no effort in this work intended to propose a specific design for this connection since, as indicated above, we are not going to deal with errors in this part of the circuit. Nevertheless, the work we hereby present may be applied in order to avoid the propagation to the network of errors which originated in faulty processors, providing both the processor and the CAN controller are integrated in the same circuit. Therefore, the triplication of processor-controller sets would also provide tolerance to faults in the processors in this specific case.

In our redundant architecture, which is shown in Fig. 2, an error (manifestation of a controller fault) manifests as a discrepancy in the value issued by each controller in its corresponding output Tx_i. In the rest of this paper, this specific error in the CAN controllers will be called *discrepancy* for short. Moreover this term is useful to distinguish this kind of error —the *processing* [11] of which is the main interest of our work— from CAN channel errors, which are already detected and tolerated by the CAN protocol.

The *Redundancy Manager* (RM) appearing in Fig. 2 is the key component of our approach. This circuit will perform all the functions related to the redundancy including typical fault tolerance functions, such as error detection, as well as other functions, such as coordinating the operation of the redundant controllers even in the absence of faults. Unfortunately, in our case it is not advisable to use a simple *passive redundancy technique* [12], such as *Triple Modular Redundancy* (TMR), to perform *error compensation* [11], i.e. *fault masking*, and, as a consequence, RM will not act as a simple majority voter on the Tx_i values. The reason why this is so will be explained in the next section.

A. Error processing by the Redundancy Manager

An RM specifically designed to adapt to CAN is required. CAN controllers, even when transmitting, monitor the channel and it is therefore fundamental that our three CAN controllers have a consistent view of said channel in order for them to simultaneously issue the same outputs (Tx_i) in the absence of faults. This is achieved by simply connecting together all reception signals (Rx_i) to the RRx signal received from the channel. Then the RM will assure that $Rx_1 = Rx_2 = Rx_3 = RRx$.

The relation between the output RTx and the inputs Tx_i is more complex. As indicated above, the RM cannot be a simple majority voter, although error detection is performed by majority voting. This means that in the event of a *discrepancy* the value sent to the channel by the majority of the controllers will be accepted as correct. Therefore the controller sending the "wrong" value will be considered as faulty and labeled as *discrepant*. Said action is in agreement with the typical assumption which states that faults affecting redundant components are independent of each other. In our case, this would be equivalent to assuming that among the set of initially working redundant controllers, the majority must still be working as the probability of simultaneous multiple faults is very low. Moreover this assumption may be strengthened during the implementation process if *common causes of faults* [11] are eliminated.

The main difference between a normal majority voter and our RM is that in the former, *error processing* is performed by *error compensation*, i.e. it takes advantage of the redundancy

contained in the erroneous state to deliver an error-free service; while in the latter it is done by *error recovery*, i.e. an error-free state is substituted for the erroneous state [11]. Simple error compensation by majority voting is not suitable in our architecture, because the complex fault tolerance mechanisms that CAN controllers perform would produce a fast attrition of the redundancy. One of those mechanisms is that a CAN controller continuously checks if the value received from the channel conforms with the value it is sending to the channel at that moment (Tx_i). For instance, if a controller is sending a dominant value to the channel but receives a recessive value, it will interpret this situation as constituting an error and will then initiate the transmission of an error flag in the following bit. Then, in the event of a transient *discrepancy* in our triplicated controller, if the discrepant controller was issuing a dominant bit, a majority voter would send a recessive bit to the channel. This value would be interpreted as an error by the discrepant controller which would then signal an error using an error flag and, later on, retransmit the frame, whereas the rest of the controllers would have already sent said frame correctly on the first attempt. Then the redundant controllers would have lost their coordinated operation forever. This kind of behaviour is equivalent to that of a TMR CAN controller, and would be correct only in case it were acceptable that a single transient fault should render the faulty controller permanently unavailable, which is a policy that leads to a fast attrition of redundancy.

Our RM avoids a fast attrition of redundancy by allowing controllers affected by transient faults to recover and restore the coordinated operation that has been lost because of the discrepancy. The simplest way of doing this in terms of hardware —without modifying the processor's software— is to take advantage of the *backward error recovery* [11] mechanism already provided by CAN. In the event of a discrepancy, RM calls this mechanism into action by directly injecting the RTx signal with an error flag visible to all the controllers of the "trio". They will then react according to the CAN protocol, interrupting the transmission of the current frame, restoring the coordination between them, and trying to retransmit the frame when the bus is idle.

Due to the majority voting carried out by RM, whenever there is a discrepancy among CAN controllers, the discrepant controller will receive through the channel a value different from that which the controller itself is transmitting. As the discrepant controller should react by transmitting an error flag, this may lead us to think that it is not necessary that the RM injects the error flag, provided that the propagation to the channel of the error flags issued by any controller is assured. Nevertheless there are several reasons which make it inadvisable that RM should rely on CAN controller's error detection and signalling mechanisms. First of all because if the controller which is supposed to detect and signal an error should itself be seriously damaged it may fail to carry out this function. Secondly because there are some scenarios in which a difference between the transmitted value and the received value may not be interpreted as a transmission error. Specifically, if during the arbitration phase a node transmits a recessive bit and receives a dominant bit this is not interpreted as constituting an error but a loss of arbitration. For all of the above reasons it is fundamental that the RM be able to inject its own error flags.

RM could initiate the injection of the error flag in the same bit in which the discrepancy is detected. Nevertheless if said injection was, in fact, initiated in that bit an undesirable situation might ensue causing non-discrepant controllers (i.e. non-faulty) to detect the error before the faulty controller itself does, for they could receive a value different from the transmitted one (as would happen, for example, if the faulty controller transmitted a dominant bit whereas the non-faulty ones transmitted a recessive bit). This situation must be prevented in order to avoid that non-faulty controllers erroneously consider themselves, according to CAN specification [1], to be the causes of the error and consequently increment their internal error counters more than the faulty controller itself. In order to prevent this situation from happening it will suffice if RM initiates the injection of the error in the bit following after the discrepancy. An additional advantage of proceeding in this manner is that the response time of RM is then a less critical parameter.

This injection of an error flag by RM means that a discrepancy caused by a triplicated controller will disturb the whole bus, and thus lead to a bandwidth reduction. Note that coordination in the triplicated controller could have been reestablished locally by the affected node by first disabling the transceiver, then using error signalling, and finally enabling the transceiver again once the coordination in the triplicated controller has been reestablished. Although this alternative approach does not disturb other nodes in the event of a discrepancy, it presents a definite disadvantage: the node affected by the discrepancy stops receiving the frames transmitted through the bus during the time its transceiver is disabled. This violates the *data consistency* property claimed by the specification of CAN protocol [1] that is fundamental in critical applications [3].

If during normal operation RM outputs to RTx the majority value, after the error flag injection has started, RM must allow any dominant value (whether it is issued by one of the controllers or by RM itself) to be propagated to RTx. This is necessary so as to adapt to the normal operation of CAN during the error signalling, which is fundamental in order to restore the coordination among all the controllers. Once RM is in injection mode it will only go back to normal mode when all controllers send recessive bits after error flag signalling and before data frame retransmission. Note that a faulty controller permanently sending dominant bits to the channel (*stuck-at-dominant failure* [4]) could block the RM while it is in the injection mode, and could then indefinitely disturb the channel. In order to avoid this behaviour, RM provides a specific counter for each controller, which ensures that the amount of consecutive dominant bits issued by each controller does not exceed the maximum value permitted by the CAN specification [1]. Should any of said counters reach this maximum, RM simply goes back to the normal mode for discrepancy detection.

B. Fault treatment by the Redundancy Manager

Besides the actions performed by RM for error processing that have been described above, RM also performs actions related to *fault treatment*, which are classically considered as constituting the second phase of fault tolerance. Fault treatment's objective is to prevent faults from being activated again. It is typically performed in two steps: *fault diagnosis* and *fault pas-*

sivation [11]. In the fault diagnosis step RM decides whether a controller is to be declared as permanently faulty or not. In order to ensure said task is accurately carried out, a *discrepancy counter* for each controller has been included in the RM design. Each time a discrepancy is detected, the discrepancy counter corresponding to the discrepant controller is increased in a predetermined amount. Moreover, each time a complete data frame is successfully transmitted, the discrepancy counters of all three controllers are decreased in a smaller amount. When one of these discrepancy counters reaches a maximum value, the corresponding controller is declared as permanently faulty.

In the second step of fault treatment, fault passivation, RM simply disables the contribution of the controller declared as permanently faulty in the fault diagnosis step. The first time this happens the triplicated controller becomes a duplicated one. While in this state, all the RM fault tolerance functions continue to be performed as before with some small differences in the event of a discrepancy between both controllers. First, the value transmitted to the channel during the discrepant bit will no longer be relevant and, as a consequence, it has been considered a "don't care" value in the implementation process. Second, there is no way to decide which controller has caused the discrepancy. Therefore RM will select that counter which showed the maximum value at the moment the other controller was disabled, and will increase it. Of course, when said counter reaches its maximum value, the whole node is disconnected from the network.

III. Design and Implementation of the RM

The design of a circuit for the RM specification introduced in the previous section has lead to a relatively simple sequential system. The design has been created assuming that CAN controllers identify a dominant bit with a logical '0' and a recessive bit with a logical '1'. The schematic for the circuit is shown in Fig. 3.

As was indicated in Section II-A, the RRx signal received from the CAN channel is connected to all reception signals (Rx_i) in order to guarantee that all the local controllers have the same view of the channel. On the other hand, operations of a greater complexity will be performed in order to obtain the value of the RTx signal which is to be output to the channel. First of all we need a circuit to generate the clock signals for the rest of the RM. This circuit, which is called *CAN Clock Unit*, generates three clock signals from the channel (RRx) following the specification provided by CAN protocol for bit synchronization [1]. Among those clock signals, Sck indicates which is the correct moment for sampling the values both at RRx and at the outputs of the local controllers (Tx_i), i.e. the instant at which they will be sampled by the other nodes according to CAN protocol. Tck indicates the instant at which the issue of the next bit starts. Rck indicates the instant at which the sampled values are available and ready for processing. Said instant is to be found situated shortly after the sampling instant indicated by Sck, and before the instant indicated by Tck.

The second circuit shown in Fig. 3 is the *Sampling Unit*. Its aim is to sample and latch the values at RRx and all Tx_i at the precise moments indicated by the Sck signal. These sampled values are called RRs, Ts1, Ts2 and Ts3, respectively. Three different circuits receive the sampled values as inputs: the *Discrepancy Detection Unit*, the *Frame Unit* and the *Fault Treatment Unit*. The Discrepancy Detection Unit is a combinational circuit which generates a series of outputs used to indicate different discrepancies between the sampled values of the Tx_i signals. D1 indicates that Ts2 and Ts3 are equal, but different from Ts1. D23 indicates that Ts2 and Ts3 have different values, a condition that is relevant once controller 1 has effectively been declared as permanently faulty and its contribution to RTx has been disabled. The rest of the outputs from the Discrepancy Detection Unit follow the same rationale. In contrast, the Frame Unit is a sequential system. It activates the Efs signal to mark the end of a frame by detecting the *end of frame sequence* of the CAN protocol at the bit stream sampled from the channel. The Fault Treatment Unit is a sequential circuit, that constitutes the kernel of RM. Said circuit is made up of a discrepancy counter and a counter of consecutive dominant bits for each redundant controller, and a state machine. This state machine decides whether the system is in the normal or in the injection mode and which controllers are disabled. When this state machine enters the injection mode, it activates the Ine signal. This circuit is also prepared to disable the contribution of the i controller by activating the corresponding Di_i signal.

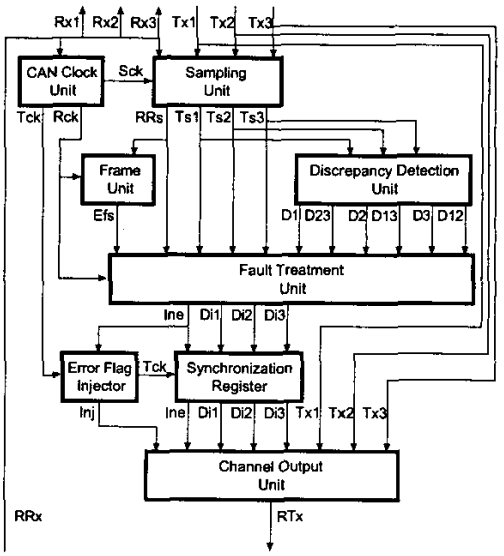

Fig. 3. The RM circuit schematic

The signals issued by the Fault Treatment Unit to disable the contributions of the different controllers to RTx (Di_i) must be allowed to change exclusively at the beginning of the transmission of the next bit. This is so because otherwise they may generate spurious edges in RTx in the middle of a time bit. Such a circumstance could mislead the bit synchronization mechanism [1] provided by CAN in the nodes of the network, since this mechanism perceives edges as indicative of the beginning of a new bit in the stream, and uses them to trigger the resynchronization process. To impose this restriction on the Di_i timing, the *Synchronization Register* is included in our RM. This register

simply latches the value of Di_i at the moment specified by the Tck signal.

The *Error Flag Injector* is a simple sequential circuit which generates the sequence of six consecutive dominant bits that constitutes an error flag. The Fault Treatment Unit enables the injector in the event of a discrepancy. The injector issues the six dominant bits through the Inj output at the precise moments indicated by the Tck signal.

The last circuit of the RM is the *Channel Output Unit*. This is the combinational circuit responsible for evaluating the definite value to be sent to the channel through RTx. The Channel Output Unit depends on the values of all its inputs as shown by the following expression

$$\begin{aligned} RTx \;=\; & (Tx_1 \vee Di_1 \vee Tx_2 \vee Di_2) \wedge (Tx_1 \vee Di_1 \vee Tx_3 \vee Di_3) \\ \wedge\; & (Tx_2 \vee Di_2 \vee Tx_3 \vee Di_3) \wedge (Inj \vee \overline{Ine}) \wedge (Tx_1 \vee Di_1 \vee \overline{Ine}) \\ \wedge\; & (Tx_2 \vee Di_2 \vee \overline{Ine}) \wedge (Tx_3 \vee Di_3 \vee \overline{Ine}) \end{aligned} \quad (1)$$

where the first three terms implement a simple majority vote, the fourth term is responsible for the effective injection of the error flags, and the last three terms allow propagation to the channel of any dominant bit issued by any local controller during the error flag injection phase.

The circuit has been implemented using the Max+Plus II tool by Altera for VHDL design and simulation. This tool allows synthesis of the VHDL code for specific FPGA circuits of the Altera family. Thus the simulation has been performed with realistic timing parameters for the FLEX10K20 FPGA. In the next section some simulation results are shown.

IV. Simulation Results

We began the simulation process with the identification of the different scenarios in which a discrepancy between the controllers may happen. Significant scenarios that have been successfully simulated include those in which there are no discrepancies, as well as those in which there are transient or permanent discrepancies whether they occur in the bits of the identifier field or in the bits which are outside it. Moreover similar cases have been considered in which the initial state of RM was that one of the controllers was disabled. We hereby give, below, a detailed description of the system's simulated response, in the light of two specific scenarios.

In the first of these scenarios, the node is transmitting a frame and, once the identifier of the message has been properly transmitted, one of the controllers produces a discrepancy by transmitting a recessive bit (logical '1') while the other two controllers transmit dominant bits. In Fig. 4 the simulator results for this scenario are shown. In this figure, among all the signals of the system only a reduced set appears. This set is the outputs of the controllers (Tx_i), the output of the RM (RTx) and both the Fault Treatment Unit and the Error Flag Injector states (FTU_state and EFI_state).

In the period labeled (A) the three controllers are transmitting the bits of the frame without discrepancy and RM is in the normal mode. At the beginning of (B) the third controller transmits a recessive bit while the others transmit a dominant bit. The value of TRx is dominant for this bit since a majority voting is performed by RM. For the next bit RM adopts the injection

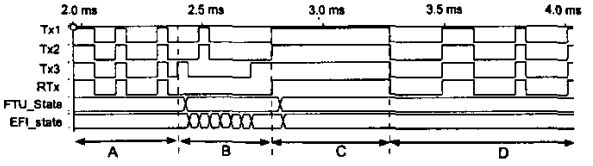

Fig. 4. Simulation results (I)

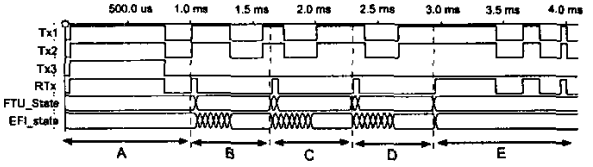

Fig. 5. Simulation results (II)

mode and initiates the transmission of an error flag. The third controller, which has registered a dominant value at the channel while transmitting a recessive value, interprets this discrepancy as a network error and starts transmitting another error flag beginning at the same bit as the RM's. The other two controllers only detect the situation one bit later while trying to send a recessive bit. At this point the value of RTx is dominant due to the error flag that is being transmitted. Controllers 1 and 2 interpret this as an error and also initiate the transmission of an error flag two bits later than the RM's. In (C) all the error flags have been transmitted and all controllers transmit the error delimiter. RM goes back to normal mode. Finally, in (D) all controllers are synchronized again and the transmission of a new data frame starts.

In the second scenario we shall analyze, the node is also transmitting and one of the controllers starts behaving differently from the rest. In this case, the third controller experiences a stuck-at-dominant failure during the transmission of the identifier field. In Fig. 5 the simulator results for this scenario are shown.

In the period labeled (A) the three controllers are transmitting without discrepancy and RM is in the normal mode. At the beginning of (B) the third controller gets stuck at dominant while the others transmit recessive bits. The value of TRx is recessive for the first bit of (B) since a majority voting is performed by the RM. For the next bit, RM changes to injection mode increasing the discrepancy counter of the third controller and initiating the transmission of an error flag. The first and second controllers interpret the first bit of the error flag as if they had lost the arbitration and they go back to the reception state. Later on within (B) the sixth bit of the error flag violates the bit stuffing rule of CAN and the first and second controllers simultaneously initiate the transmission of their own error flags. By the end of (B) all the error flags have been transmitted but instead of having a recessive value at RTx, and as RM is in the injection mode, the dominant value from the third controller is propagated to RTx. The counter of the Fault Treatment Unit intended to count the number of consecutive dominant bits reaches its maximum and RM then goes back to normal mode. Once in (C) RM detects a new discrepancy, increases the discrepancy counter of the same

controller, and goes back to injection mode initiating the transmission of a new error flag. The first and second controllers interpret the first bit of the error flag as a violation of the error frame format since, according to CAN, all controllers should be sending seven consecutive recessive bits and consequently begin transmitting their own error flags. At the end of said transmission the situation will be such as it was at the end of (B): the dominant bit from the third controller is propagated to RTx, the counter of consecutive dominant bits reaches its maximum, and RM goes back to normal mode. The behaviour of RM while in (D) is identical to that which it exhibits while in (C), with a single difference; by the end of (D) before going back to normal mode, the discrepancy counter of the third controller has reached its maximum. Therefore this controller is disabled when the RM switches to normal mode. So in (E) the system continues working with only two controllers.

V. Conclusions and Future Work

In this paper, a fault-tolerant CAN controller subsystem has been proposed. This subsystem is made up of three standard CAN controllers and a specifically designed circuit, which manages the given redundancy. The complete architecture of this redundancy management circuit, called RM, has been explained. RM has been described in VHDL and successfully simulated using the Max+Plus II tool by Altera. Some of these simulations have been presented.

An important advantage of our CAN controller subsystem is that it achieves tolerance to CAN controller faults without modifying the rest of the system, e.g. the application software. This specific feature opens room for the integration of the RM in CAN nodes that already present other fault tolerance mechanisms, e.g. network media redundancy [4]. RM performs error processing and fault treatment tasks specifically adapted to the CAN protocol characteristics. In the absence of faults, the resultant fault-tolerant CAN controller works as a standard one, e.g. without any communication overhead. Moreover it can tolerate the permanent failure of one of the controllers and any number of transient faults, in this last case, without redundancy attrition. Both in the absence and in the presence of faults at the controllers, our approach preserves the data consistency property of CAN. It is important to note that although the RM represents a single point of failure, we have tried to keep it as simple as possible in order for it to be highly reliable in comparison with CAN controllers.

Further work addressing the problem of controller fault tolerance in CAN nodes can be found in the literature [5], [6]. Nevertheless, none of these papers suggest a solution to this problem at the controller level. More in detail, they require important modifications of the application software executed by the node's processors. The solutions proposed in that papers present additional disadvantages. On the one hand, the architecture presented in [5] is only able to detect controller faults as long as they manifest as channel errors which are detectable by CAN as well as those faults which produce a crash failure. On the other hand, the architecture in [6] requires four controllers in order to be able to tolerate the permanent failure of one of them.

Future work will include the design of a circuit to manage the redundant communication between the triplicated controller and the processor and to impede the controller errors to affect the processor. Future work will also include the modelling of the complete redundant controller in order to assess the reliability achieved.

VI. Acknowledgments

Our present paper has been supported by the spanish MCYT grant DPI2001-2311-C03-02, which is partially funded by the European Union FEDER program. The authors wish to thank Alberto Ortiz for his help in the processing of the text and the figures. Special thanks to Astrid Aguayo, language consultant.

References

[1] ISO. *International Standard 11898 - Road vehicles - Interchange of digital information - Controller Area Network (CAN) for high-speed communication*. 1993.

[2] K. Tindell and A. Burns. Guaranteeing message latencies on control area network (CAN). Technical report, University of York, Department of Computer Science, 1995.

[3] J. Rufino, P. Veríssimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-tolerant broadcast in CAN. In *Proc. IEEE 28th Int. Symp. Fault-Tolerant Computing. FTCS'98. Munich (Germany)*, June 1998.

[4] J. Rufino, P. Veríssimo, and G. Arroz. A Columbus' egg idea for CAN media redundancy. In *Digest of Papers, The 29th Int. Symp. on Fault-Tolerant Computing. FTCS'99. Madison, Wisconsin, USA*, pages 286–293, June 1999.

[5] H. Hilmer, H.-D. Kochs, and E. Dittmar. A fault-tolerant communication architecture for real-time control systems. In *Proc. IEEE Int. Workshop on Factory Communication Systems. Barcelona (Spain)*, October 1997.

[6] J. Proenza, J. Pons, and J. Miro-Julia. A low-cost fail-safe circuit for fault-tolerant control systems. In *6th IEEE Int. Conf. on Electronics, Circuits and Systems . ICECS'99. Pafos, Cyprus*, September 1999.

[7] I. Broster and A. Burns. Timely use of the CAN protocol in critical hard real-time systems with faults. In *Proceedings of the 13th Euromicro Conf. on Real-time Systems*, 2001.

[8] T. Führer, B. Müller, W. Dieterle, F. Hartwich, R. Hugel, and M. Walther. Time triggered communication on CAN (Time Triggered CAN- TTCAN). In *7th International CAN Conference. ICC'00. Amsterdam. Netherlands*, October 2000.

[9] H. Kopetz and G. Grünsteidl. TTP—a protocol for fault-tolerant real-time systems. *IEEE Computer*, 27(1):14–23, January 1994.

[10] C. Guerrero, G. Rodríguez-Navas, and J. Proenza. Hardware support for fault tolerance in triple redundant can controllers. In *9th IEEE Int. Conf. on Electronics, Circuits and Systems . ICECS'02. Dubrovnik, Croatia*, September 2002.

[11] J.-C. Laprie, editor. *Dependability: Basic Concepts and Terminology*. Springer-Verlag Wien New York, 1992.

[12] B. W. Johnson. *Design and analysis of fault tolerant digital systems*. Addison-Wesley Publishing Company, 1989.