An active star topology for improving fault confinement in CAN networks

Manuel Barranco, Julián Proenza, Guillermo Rodríguez-Navas and Luís Almeida, *Member, IEEE,*

***Abstract*—The Controller Area Network (CAN) is a field bus that is nowadays widespread in distributed embedded systems due to its electrical robustness, low price, and deterministic access delay. However, its use in safety-critical applications has been controversial due to dependability limitations, such as those arising from its bus topology. In particular, in a CAN bus there are multiple components such that if any of them is faulty, a general failure of the communication system may happen. In this paper, we propose a design for an active star topology called CANcentrate. Our design solves the limitations indicated above by means of an active hub which prevents error propagation from any of its ports to the others. Due to the specific characteristics of this hub, CANcentrate is fully compatible with existing CAN controllers. The paper compares bus and star topologies, analyzes related work, describes the CANcentrate basics, paying special attention to the mechanisms used for detecting faulty ports, and finally describes the implementation and test of a CANcentrate prototype[1].**

***Index Terms*—Fault tolerance, system analysis and design, communication system fault tolerance, communication system fault diagnosis, field buses, CAN protocol, star, topology, hub.**

I. Introduction

The Controller Area Network (CAN) protocol is a field bus which fulfills the communication requirements of many distributed embedded systems. In particular, CAN provides high reliability and good real-time performance with very low cost. Due to this, CAN is nowadays used in a wide range of applications, such as factory automation or in-vehicle communication. Furthermore, there is still a high interest in researching new solutions based on the mentioned properties of CAN [1] [2].

Nevertheless, communication systems based on CAN present several specific dependability problems, some of which are caused by their bus topology. The main drawback of any protocol using a bus topology is that the structure of the network presents multiple components, which have direct electrical connections to each other without proper error containment. As a consequence, a fault in any of them may generate errors that propagate and effectively prevent further communication to take place. An example of this situation is depicted in case A of Figure 1, in which a fault in the medium access circuitry of node 2, e.g. with the transmitted bits stuck at a fixed value (a dominant value in case of CAN), blocks the communication channel and none of the nodes can communicate with each other. Similar situations can happen

[1]The contents of this article have been the subject of a patent filing submitted on the 16th of September of 2004.

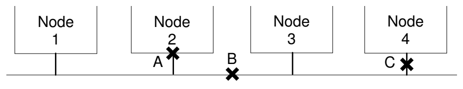


Fig. 1. Examples of failures of the communication system

with faults in the node itself, as well as with short circuits in the bus transmission medium or in the connectors.

Moreover, a bus is shared by all communication paths between every subset of nodes. Consequently, a partition in just one point necessarily leads to a disruption of many communication paths. This is depicted in case B of Figure 1 in which a partition in the bus mid point blocks any further communication between nodes 1 and 2 with nodes 3 and 4.

Finally, case C shows the situation in which there is a partition in the local connection of node 4 with the bus that does not affect the bus integrity and which leaves node 4 isolated. Consequently, the communication among the remaining nodes is unaffected. From the communication system point of view, this is the desired behavior when a fault occurs in one node or node bus interface, because it exhibits the least impact on the communication system itself.

The general framework within which this work has been developed addresses two main objectives. The first objective is to prevent situations in which one single fault in the communication system leads to a *severe failure of communication*, e.g. cases A and B in Figure 1. For the purpose of this paper, we define *severe failures of communication* as those in which more than one node cannot communicate with two or more of the other nodes. In practice, we achieve this objective by using an appropriate topology, namely a star, provided with an innovative hub that enforces the necessary error containment. This hub blocks error propagation from faulty nodes or from faulty links to the rest to the system at the respective hub port.

Nevertheless, the star topology still contains one single point of failure, i.e. the hub, which if faulty may lead to a global communication failure. Even so, we consider the star topology to be a good choice because it is easier to improve dependability for the unique single point of failure of the star, e.g. the hub can be placed in a more protected area within the system, than for the multiple components that may cause a severe failure of communication in a bus topology. Moreover, replicated star topologies can be used to tolerate both hub as well as link faults.

The second objective of our general framework is to exploit the possibilities the star topology offers to further improve dependability of CAN networks. In particular, the hub can be provided with the appropriate capabilities to mitigate the impact of faults not included in the scope of the first objective, i.e. those which do not lead to a severe failure of communication. For instance, the hub could prevent that any node impersonates another node, thus restricting the failure semantics of the nodes.

This paper is devoted to the first objective, only, and it addresses the design and implementation of a new simplex star topology we have called CANcentrate. The design of a replication scheme for CANcentrate can be found in our recent work [3], whereas the achievement of the second objective pointed out above will be the subject of future work.

One requirement was imposed from the beginning on the design of CANcentrate: to preserve all the specifications of CAN [4]. As a consequence of this compatibility, CANcentrate preserves all the dependability properties already achieved by CAN; commercial off-the-shelf components can be used for building the CAN nodes of CANcentrate; and the hub is transparent for any application that uses CAN.

In the following section we discuss the properties of existing solutions to improve the dependability properties of CAN, focusing on the advantages of a simplex star topology with respect to simplex and replicated bus topologies. Moreover, existing work on star topologies for CAN is also presented. Section III presents the architecture of CANcentrate. Section IV discusses the mechanisms that the hub of CANcentrate includes in order to diagnose faulty nodes or links and disconnect them from the network. A prototype implementation of CANcentrate is described in Section V. Section VI summarizes the paper.

II. Problem statement

Despite the good dependability properties exhibited by the CAN protocol, it still presents some drawbacks. The several techniques previously proposed in the literature to confine the effect of faults in CAN systems are not completely effective. This is because they still allow single faults of different types and occurring in different physical points to provoke a severe failure of communication.

Therefore, we can rewrite the objective of this work as preventing the existence of multiple components in a CAN network such that a single fault in any of them may cause a severe failure of communication.

In order to better understand how CANcentrate achieves this objective, next we describe the fault model and fault assumptions considered in this work.

A. Fault model and assumptions

The fault model gathers the different kinds of faults that may happen in the components of a CAN network and that may cause a severe failure of communication:

- *Stuck-at node* fault. It occurs whenever a given node is damaged and issues a constant bit value. Two types of stuck-at fault exist: stuck-at-dominant and stuck-at-recessive faults, depending on whether the bit value issued by the faulty node has dominant or recessive value respectively. In CAN, the logical '0' value is referred as the dominant value, whereas the logical '1' is referred as the recessive. Therefore, since the physical layer of CAN is equivalent to a logic-AND of every node's contribution, only a stuck-at-dominant fault may cause a severe failure of communication.
- *Shorted medium* fault. This occurs whenever the medium is electrically connected to battery or to ground due to a short-circuit. For obvious reasons, this fault prevents any communication. When fault-tolerant cabling is used, as recommended in CAN [4], such a fault only happens when both wires are shorted to a fixed low impedance electrical source.
- *Medium partition* fault. It occurs whenever the medium is interrupted in such a way that the network is broken into several subnetworks, which are called *network partitions*. Therefore, any two nodes which are each one in a different partition can no longer communicate with each other. Moreover, signal reflections at the open extremities may cause channel errors that prevent nodes in the same partition from communicating properly [4].
- *Bit-flipping* fault. This occurs whenever a component of the network (either a node or a medium) exhibits a *fail-uncontrolled* behavior and starts sending erroneous and random bits with no restrictions in the value domain. In this case, even if a node is trying to send a correct bit stream, this is destroyed by the dominant bits of the bit-flipping stream. Some potential causes of this fault are: a damaged node that sends random bit values; a bad welding on the medium connector that generates random bit values, etc.
- *Babbling-idiot* fault. It occurs whenever a node sends messages that are erroneous in the time domain, then consuming more resources that it really needs and starving the other nodes of the appropriate resources for communicating [5]. For instance, this type of faults may happen as a consequence of a software fault in a node that results in an infinite loop that sends messages continuously. To deal with this kind of faults requires knowledge about the scheduling of the messages, which depends on the application executed at nodes. However, in the context of this present work we focus on a solution independent of the application. Therefore, although a babbling-idiot fault may provoke a severe failure of the communication system, we postpone its treatment for future work. Note that, for instance, it is suitable to include in the hub a bus-guardian, similar to the one proposed in [5], for dealing with faults in the time domain.

Beyond the types of faults considered in this work as stated above, we make no assumptions on the frequency and duration of errors that may occur in any port or group of ports of the hub. The only assumption made, since hub replication is not considered, is that the hub itself will not fail.

B. Potential solutions

Some of the faults presented above can be confined in bus-based CAN systems, up to a certain extent, using techniques that are already known. These techniques are: the use of replicated transmission media such as in [6] [7], the use of reconfigurable transmission media such as in RedCAN [8], and the use of bus-guardians such as in [5].

However, due to the characteristics that are inherent to the bus topology, these techniques do not prevent the existence of multiple components such that a single fault in any of them may cause a severe failure of communication. First, replicated buses may suffer from *common-mode* failures due to the fact that they come together near every node, i.e. due to spatial proximity [9]. Moreover, replicated buses cannot prevent a faulty node from causing a severe failure when the node sends incorrect information to all media. Second, RedCAN connects nodes by means of a special ring in which one of its sectors is redundant and only becomes active for tolerating stuck-at faults in other sectors. Nevertheless, RedCAN only deals with faults occurring in adjacent sectors or contiguous nodes; requires specific RedCAN hardware for implementing nodes; and increases the complexity of the nodes, thus increasing their probability of failure. Finally, a bus-guardian can only contain the propagation of errors generated by the node it supervises and, thus, the bus-guardian is useless for containing error propagation from a faulty medium. Moreover, a bus-guardian and its corresponding node may also exhibit common mode failures, due to spatial proximity, or due to the fact that the guardian and its node can share resources or procedures, e.g. power supply, clock synchronization algorithm, etc.

Since all these techniques –even if they are used together– still allow multiple components to cause severe failures of communication, alternative solutions have been researched, namely those based on a star topology. In a star topology, each node is connected to a central element, the *hub*, by its own *link*. One advantage of a star topology is that links only come into spatial proximity at the center of the star and that a medium partition cannot cause a network partition. But the most important advantage is that the center of the star, the hub, has a privileged view of the system, as it simultaneously knows the contribution from each node through its corresponding link. Hence, an adequate hub could enforce confinement of faulty transmission media and faulty nodes, by disconnecting the adequate hub ports. Furthermore, fault independence would be ensured between the hub and the nodes, if the hub acted as their guardian.

It is obvious that the main drawback of a star topology is that the hub represents a single point of failure. Nevertheless, different strategies can be adopted in order to face this problem. For instance, the hub reliability can be increased by placing it in a well-protected zone inside the physical system or by investing in its quality or even by adopting a replicated star topology, as the one we have recently proposed in [3].

Some star topologies for CAN can be found in the literature or in the market [10] [11] [12] [13]. Some of them are passive stars in the sense that the hub acts as a concentrator where all the incoming signals are coupled. These stars present important disadvantages [14] concerning coupling loses, strong limitations on the star radius or in the bit rate, electrical problems, etc. Other types of stars are known as active stars, which overcome some technical problems of passive stars. These topologies rely on an active star coupler, which receives the incoming signals from the nodes bit by bit, implements a logical AND, and retransmits the result to all nodes. Unfortunately, neither of these passive nor of these active stars do address severe failures of communication, or only deal with stuck-at-dominant faults. Moreover, some of them are not even compatible with the CAN protocol. A deeper discussion on the drawbacks of existing passive and active star topologies can be found in [14] and [15].

III. Design of CANcentrate

In order to overcome the inability of the architectures described in the previous section to eliminate the presence of multiple points of severe failure, we have proposed a new communication infrastructure called CANcentrate.

CANcentrate is based on an active star topology and presents a new hub that includes enhanced fault-treatment mechanisms. From the hub perspective, each node together with its dedicated link constitute an error-containment region. A permanent fault within any of these regions manifests as a permanently faulty port, which can be isolated by the hub to prevent a severe failure from occurring.

This section is devoted to describing the CANcentrate architecture, paying special attention to the internal structure of the hub.

A. *CAN synchronization and error handling basics*

CAN communication relies on a complex bit synchronization mechanism which guarantees that nodes have a quasi-simultaneous view of every single bit on the channel (the so-called *in-bit response*). This mechanism uses the recessive to dominant transitions of the signal on the channel in order to keep the nodes of the network synchronized with respect to the node that is transmitting (the so-called *leading transmitter*). In this way, nodes do not only agree in each bit value, but also about the location of each bit inside the frame, i.e. nodes are synchronized at both bit-level and frame level. This synchronization allows definition of a number of additional mechanisms (e.g. bit-wise arbitration, ACK bit, error frame), which significantly improve the dependability and real-time properties of CAN networks [4].

Other important characteristics of CAN related to dependability concern the error-detection and the error-signaling mechanisms. Any CAN node is able to detect five different types of errors [4]: *stuff error*, *format error*, *bit error*, *CRC error* and *ACK error*.

Whenever a CAN node detects an error, it signals it by transmitting an *error flag* [4]. This flag aims at compelling the other nodes to detect an error too and, consequently, to abort the transmission/reception of the frame that is currently being transmitted. In other words, the transmission of an error flag upon error detection aims at enforcing that any frame is either accepted by all nodes or rejected by all. This property is called *data consistency* in the CAN specification [4].

There are different types of error flags, namely *active and passive* error flags [4]. However, only active error flags, which can only be sent by nodes that have not detected too many errors so far (and thus nodes that are assumed as not being

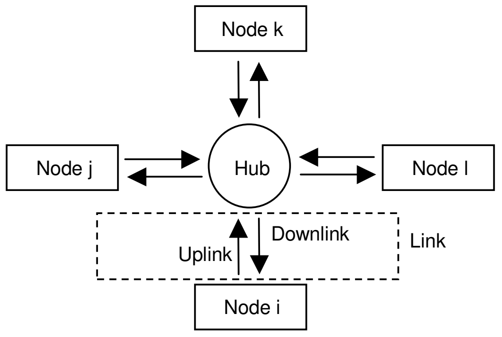


Fig. 2. Architecture of CANcentrate

affected by a permanent fault), ensure the globalization of an error and, hence, data consistency.

The ability of globalizing an error is also important because it allows nodes to re-synchronize after the detection of an error. Note that a node detecting an error must assume that it has become de-synchronized. This is because a node cannot know whether the other nodes saw the same error at the same time or not, as well as whether the error appeared as a consequence of a previous de-synchronization at frame level. In case an error flag globalizes an error, all the nodes transmit their own error flags and, then, cooperatively transmit an *error delimiter*. This error delimiter is built in such a way that all nodes finish transmitting it at the same time thus re-synchronizing at frame level again [4]. The frame constituted by all the overlapped error flags, followed by the cooperative error delimiter is called *error frame*.

B. Design rationale

Assuming dominant/recessive transmission, the hub must implement a logical AND function of the individual transmissions received from every node. Moreover, and in order to preserve the in-bit response, this logical AND must be performed within a fraction of one bit time, despite the extra delay which the internal circuitry of the hub may cause.

Furthermore, the hub must include some mechanisms in order to identify permanently faulty ports. These mechanisms require the hub to be able to discriminate the signal which any node transmits from the signal resulting of the logical AND which the hub broadcasts to the nodes. A simple way to separate both signals is through the use of two different cables for each link that connects each node to the hub. Figure 2 shows the architecture of CANcentrate. The cable which carries the signal from a node to the hub is called the *uplink*, whereas the cable which carries back the resulting signal from the hub to the node is called the *downlink*.

Therefore, two transceivers are required at the end of each link; one for the uplink and another one for the downlink. Figure 3 illustrates how the transceivers are connected at the node's end. Note that the *receive data output* pin (RxD) of the uplink transceiver is left open whereas the *transmit data input* pin (TxD) of the downlink transceiver is forced to have a recessive level (the logical '1' value). It is important to remark that the CANcentrate architecture can be implemented with both

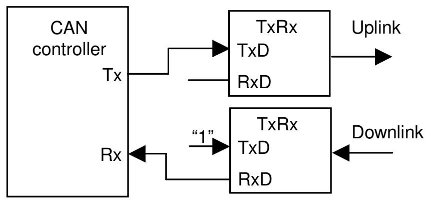


Fig. 3. Configuration of the transceivers to connect a node to its link

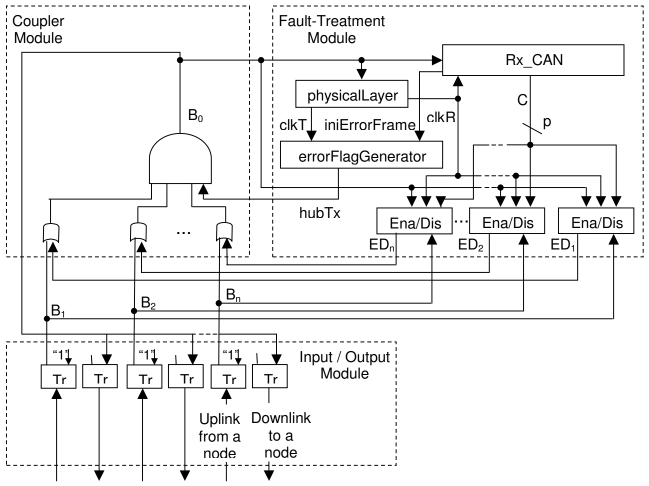


Fig. 4. Internal structure of the hub

commercial off-the-self CAN controllers and commercial off-the-shelf CAN transceivers. This makes the solution practical and relatively low-cost. Nevertheless, the hub requires some specifically designed hardware, as discussed next.

C. Internal structure of the hub

The hub is divided into three modules, namely the *Input/Output Module*, the *Coupler Module*, and the *Fault-Treatment Module*. The structure and interconnections of these modules are depicted in Figure 4.

The Input/Output Module is made up of a number of transceivers; two for each link. As Figure 4 shows, one transceiver is assigned to every uplink in order to convert the physical signal received from each node into a logical value that the hub can process, $B_{1..n}$. Moreover, one transceiver is assigned to every downlink so that the logical output of the hub, the coupled signal B_0, is converted into a physical signal that is broadcasted to every node.

The Coupler Module is made up of an AND gate, which performs the coupling of the uplink signals, and a number of OR gates, one per link, which allow the hub to disable the contribution to the global AND from a specific uplink that has been diagnosed as being permanently faulty. Since the AND gate replaces the wired-AND functionality of the CAN bus, this means that the output of the Coupler Module would be the same of a CAN bus where there were no permanently faulty component. The frame that results from coupling the frames from the enabled ports is called the *resultant frame* hereafter.

The main purpose of the Fault-Treatment Module is to detect permanently faulty ports and to isolate them from the system, so they cannot cause severe failures of communication. This function is carried out by performing both *fault diagnosis*, which aims at finding out permanently faulty ports, together with *fault passivation*, which aims at isolating permanently faulty ports from the system.

The fault-diagnosis mechanisms of the Fault-Treatment Module require the identification of the contributions from every uplink as well as knowledge of the *current state* of the *resultant frame*.

Fortunately, the use of a dedicated link for each node, as well as the separation between uplinks and downlinks, allow the identification of the contribution sent by each node.

The current state of the *resultant frame* represents what all nodes are supposed to have received from the hub until this moment. Therefore, it permits to identify which is the meaning of the bit of the *resultant frame* that is currently being broadcasted to all ports, as well as to forecast which should be the proper contribution of each node for the following bit.

The knowledge of the current state of the *resultant frame* requires to keep the synchronization of the hub with the *resultant frame* at bit level as well as at frame level. On the one hand, the *Physical Layer Module* in Figure 4 uses the typical CAN synchronization mechanisms [4] for allowing the hub to synchronize with the bit stream at bit level. In particular, this mechanisms generate the reception and the transmission clock signals (clkR and clkT) that indicate when to sample each bit value and when to issue a bit to the medium, respectively.

On the other hand, the *Rx_CAN Module* observes the bit stream at the coupled signal B_0 in order to achieve the synchronization at frame level. As a result of this synchronization, it generates a set of signals, C. These signals together with B_0 describe the current state of the *resultant frame* that will be used for fault diagnosis by the Enabling/Disabling units (Ena/Dis in Figure 4). The set of signals C provide the following information: whether the bit is a stuff bit, the expected correct bit value according to the stuff rule, the type of frame and the specific frame field the bit belongs to, whether the bit is the last bit of the End-Of-Frame field, and whether the frame has passed the CRC checking (see Figure 5).

In order to keep the synchronization at frame level with the nodes in spite of the presence of errors, when the hub detects an error in the resultant frame, it forces a re-synchronization by means of transmitting an active error flag. Since the hub is not the original transmitter of the message, it can only detect in the *resultant frame* those errors that a CAN node would detect while receiving: stuff error, format error and CRC error. When the Rx_CAN Module detects any of these errors at the *resultant frame*, it compels, via the *iniErrorFrame* signal, the *Error Flag Generator Module* (*errorFlagGenerator* in Figure 4) to transmit an active error flag through a dedicated contribution, *hubTx*, driven into the global AND.

The final *fault diagnosis* and *fault passivation* are carried out by the *Enabling/Disabling* units (*Ena/Dis* in Figure 4). Each one of these units uses the set of signals C and the coupled signal B_0 to know the current state of the *resultant frame*. This information is used together with the contribution from its corresponding port (either B_1, B_2, etc.) in order to diagnose whether its port is permanently faulty or not.

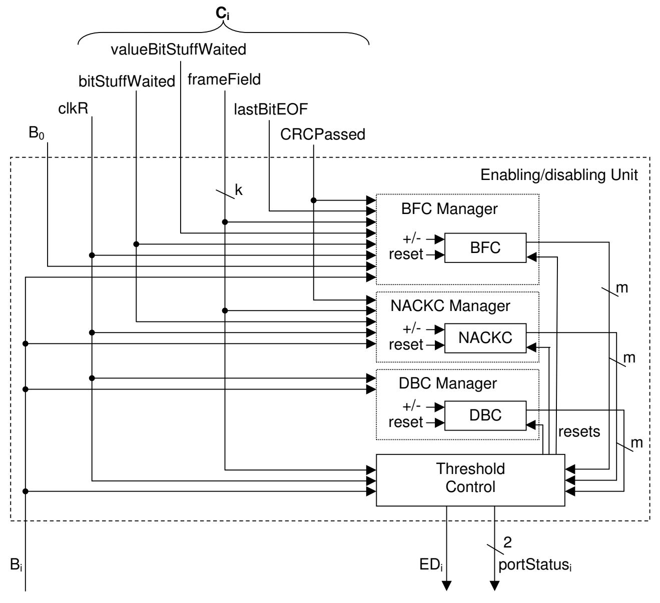


Fig. 5. Internals of one Enabling/Disabling Unit

Whenever a given Enabling/Disabling Unit diagnoses its corresponding hub port as being permanently faulty, it removes the contribution of this port from the system by issuing a logical '1' to the corresponding *Enabling/Disabling* signal, $ED_{1..n}$, which is connected to the OR gate that corresponds to the faulty port (see Figure 4). In general, this mechanism is similar to the one proposed in [6] to manage, locally in each node, the media redundancy in a replicated bus topology.

IV. Fault-diagnosis mechanisms

The main objective of the fault-diagnosis mechanisms the hub includes is to detect permanently faulty ports. The failures that are diagnosed by the Fault-Treatment Module cover all the faults that are in the scope of this present work: stuck-at-dominant, stuck-at-recessive and bit-flipping faults. Notice that a shorted medium manifests itself at the hub port as a stuck-at fault, whereas a medium partition can manifest itself as either a stuck-at fault or a bit-flipping fault. That is because a shorted medium is stuck at a constant voltage level (battery or ground), whereas a medium partition can either force the medium to be stuck at a constant voltage level, or cause channel errors due to signal reflections at the open extremities of the cable.

The fault-diagnosis mechanisms are essentially implemented by one Enabling/Disabling Unit for each port, whose internals are shown in the Figure 5. These units operate separately on each port, thus diagnosing a port as permanently faulty with high accuracy and reducing the probability of isolating non-faulty ports.

On the one hand, each Enabling/Disabling Unit has a dedicated *event counter* and an associated *manager module* for each type of fault that must be detected, as will be explained later. Each management module basically analyzes the coupled signal, B_0, its corresponding port contribution B_i and the state signals C, from Rx_CAN, in order to decide how to increase or decrease its corresponding event counter.

On the other hand, each Enabling/Disabling Unit has a *Threshold Control Module* that is aimed at declaring the port as permanently faulty when appropriate. The Threshold Control Module takes into account the value registered by each event counter and is programmed with a specific threshold for each of them. Whenever any of the event counters exceeds its corresponding threshold, the Threshold Control Module declares the corresponding port as permanently faulty and isolates the port contribution by setting the corresponding ED_i signal to '1'. However, in order to increase the tolerance to transient faults, the Threshold Control Module may use a specific reintegration policy to re-enable the port contribution and to allow the operation of all managers again, after a given period of *inactivity* is observed at the port. This reintegration policy has been described in [14].

A. Stuck-at-dominant faults

In order to detect stuck-at-dominant faults, each Ena/Dis unit includes an event counter called *Dominant Bit Counter* (DBC) and a *DBC Manager Module*. A DBC counts the number of consecutive dominant bits that are received from its corresponding port. The DBC Manager increases the DBC in one unit or resets the DBC when observes a dominant or a recessive bit at the uplink respectively. Whenever the DBC value exceeds the *Dominant Bit Threshold* (DBT), the Threshold Control Module isolates the corresponding port.

The DBT is configured in order to maximize the chances to differentiate between situations in which a port really suffers a stuck-at-dominant fault and situations in which the port is occupied by many consecutive dominant bits, although it is not stuck-at-dominant. The value of the DBT takes into account two different contributions:

$$\text{DBT} = (T_{\textit{stuff}} + 1) + N * T_{\textit{errorFlag}}$$

The first term, $T_{\textit{stuff}} + 1$, specifies the minimum number of consecutive dominant bits that violates the stuffing rule in a CAN network (6 bits). This term includes the maximum number of consecutive dominant bits allowed in CAN data frames, $T_{\textit{stuff}}$, plus the additional dominant bit needed for violating the stuff rule. Whenever the stuff rule is violated, it is expected that all nodes start to send an active error flag. Nevertheless, it is possible that a node detects a second error during its own error flag and starts sending again an active error flag, thereby prolonging the sequence of consecutive dominant bits. In the worst case, a node will see this second error in the last bit of its first error flag, and will send a consecutive active error flag. The second term, $N * T_{\textit{errorFlag}}$, is intended to cover these situations. It specifies the maximum number of consecutive dominant bits that are considered as overlapped or consecutive active error flags.

Note that for $N = 2$ the threshold coincides with the one proposed in [6]. In that case, the threshold can be exceeded if two additional errors occur in the error flag that follows a violation of the stuff rule. Using a higher value of N reduces the probability of performing an erroneous stuck-at-dominant diagnosis.

The value of N can be configured depending on the application. For instance, in an environment with high electromagnetic interference, we may consider that $N = 4$ is tolerant enough and does not imply a significant loss of reactivity in diagnosing stuck-at-dominant faults.

B. Stuck-at-recessive faults

Due to the AND function that the hub implements, a port suffering a stuck-at-recessive fault does not interfere the communication among the rest of the nodes in the star, and cannot cause a severe failure of the communication system. Nevertheless, detection of such faults may still be useful in order to implement additional fault-tolerance mechanisms at higher levels of the system architecture, for example to detect a crashed or absent node.

Since a CAN node may be without transmitting, which actually means sending recessive values, for a long time, it would be theoretically impossible to differentiate between a stuck-at recessive node and an operational but non-transmitting node. Nevertheless, the CAN protocol specifies that every CAN controller must transmit a dominant bit in the ACK slot of every frame that is correctly received [4]. Therefore, the absence of this bit can be used to detect stuck-at-recessive ports.

For each port, such detection is carried out by a specific *Non-Acknowledge Counter* (NACKC) and its manager module, *NACKC Manager Module*. Whenever the NACKC Manager detects, thanks to the C signals, that the current state of the *resultant frame* is the ACK slot and that the frame has passed the CRC checking, it checks in B_i if the node is sending a dominant value to acknowledge the frame. If this dominant value is not sent, then the NACKC Manager increases the NACKC. The NACKC Manager decreases the NACKC whenever a dominant bit is issued through the port.

When the NACKC exceeds the *Non-Acknowledge Threshold* (NACKT), the Threshold Control Module does not disable the port, but notifies the user about the inactivity of the port by means of a LED. The specific value for the NACKT can be configured depending on how strict we want to be when considering a port as being stuck-at-recessive. For instance, since a node should send an active error flag after omitting an ACK bit, even a NACKT value equal to 2 can be considered if we want to be very strict when detecting a crashed or absent node.

C. Bit-flipping faults

The CAN standard specifies a mechanism in each node that may be used in order to detect bit-flipping faults: a *Transmission Error Counter* (TEC) and a *Reception Error Counter* (REC). These counters are increased and decreased following some rules established in the CAN specification, and may cause a node to reduce its impact on the communication process, for instance, by disconnecting itself from the network in order to prevent further propagation of local errors.

Nevertheless, the mechanism based on the TEC/REC presents some deficiencies that suggest the hub should not rely that nodes will ensure fault confinement. First, normal

CAN nodes can fail in arbitrary ways and for this reason may stop performing fault confinement. Second, if a medium is the source of bit-flipping faults affecting all nodes, it cannot be isolated by the nodes. Moreover, to replicate the TEC/REC within the hub for each port is not recommendable, either, because such mechanism was designed for a bus in which the contributions of all nodes are mixed, thus with an error-detection accuracy lower than what is achievable with the hub. Therefore, we decided to include in each Enabling/Disabling Unit a dedicated *Bit-Flipping Counter* (BFC) and its associated *BFC Manager*.

Each BFC Manager independently evaluates the correctness of its corresponding port by checking whether its contribution B_i deviates from the expected behavior according to the current state of the *resultant frame* (C and B_0). Although the behavior of a CAN node is quite complex in the general case, we have been able to identify three independent types of behaviors: when a node is a leading transmitter, when a node is a receiver and when a node is engaged in error and overload signalling [4].

A transmitter node must still respect the restrictions imposed by the CAN standard, thus the BFC Manager adapts the error-detection mechanisms specified in the CAN protocol [4]. By means of these mechanisms it directly detects errors effectively generated by the transmitter node. Notice that, in contrast, in a bus the node contributions are mixed and so an error can be assigned to the transmitter even if it was in fact generated elsewhere.

When a node is a receiver, the BFC Manager expects this node to send only recessive bits, except for two cases. First, the BFC Manager expects a dominant bit during the ACK slot if the frame observed by the hub has passed the CRC checking performed by the Rx_CAN Module. Second, the BFC Manager will allow a dominant bit during the *idle* field since it would correspond to the transmission of a *Start of Frame* bit [4].

During error or overload signalling, the BFC Manager checks whether the node correctly cooperates to construct the active error or the overload frame, including the case in which the node is the first that starts transmitting the error frame.

Finally, the BFC Manager isolates the port when the BFC exceeds a specific *Bit-Flipping Threshold* (BFT). For the specific value of the BFT and for the number of units that the BFC has to be increased or decreased, we recommend to use values such that the fault diagnosis performed by the BFC Manager is more strict than the fault diagnosis performed by a CAN node using the TEC/REC [4]. For instance, a possible solution could be to increase the BFC in 8 units when detecting any error, to decrease it in 1 unit when a data frame or a remote frame is correctly transmitted, and to set the BFT to 128.

V. CANcentrate prototype implementation

This section is aimed at describing the basics of the first prototype of CANcentrate and presenting experimental results. The prototype is divided into several parts. The internal part of the hub (the Coupler Module and the Fault-Treatment Module) has been implemented using the VHSIC Hardware Description Language (VHDL) and a Xilinx Field Programmable Gate Array (FPGA). A dedicated board was built for implementing the Input/Ouput Module with capacity to connect four nodes. One UTP (Unshielded Twisted Pair) Category 5/5e/6 cable was used for each link. Each uplink / downlink within a cable used a different two-wire differential line. The CAN nodes have been totally implemented using commercial off-the-self components, and were based on *CANivete* boards [16] with just an extra CAN transceiver.

The tests to check the functionality of the CANcentrate hub were carried out at two different levels: at the level of the VHDL design of the hub and at the level of the physical network. Under error-free conditions, the following aspects have been checked at both levels: the state machines that constitute the hub, the synchronization at bit level and at frame level, and the correct assignment of roles (transmitter/receiver) of the nodes after the arbitration phase. Additionally, many different scenarios concerning the faults included in the scope of the present work were tested. Specifically, we checked that the hub correctly increases the appropriate event counters and isolates the corresponding ports whenever the pertinent thresholds are exceeded. In every case, the observed behavior of the hub was correct during both error and error-free conditions.

For thoroughly testing CANcentrate at the level of the physical network, we built an experimental platform made up of three CAN nodes. In each node a test software was continuously trying to transmit CAN frames without any additional delay. This ensures that the network load is close to the maximum and, with 3 nodes, there is an arbitration every frame.

Furthermore, different fault-injection mechanisms were used in combination with this platform. Stuck-at-recessive faults were easily injected by disconnecting the link of a CAN node from the hub. In contrast, injection of both stuck-at-dominant and bit-flipping-faults was done by means of a signal generator device. Transmitting different periodic square signals with periods ranging from a few bit times to several frame times, allowed verifying both the stuck-at-dominant isolation and the further re-enabling of the ports. Regarding bit-flipping faults, the signal generator was used with short periods from a few bit times to less than one bit time. Additionally, bit-flipping faults were also injected by mechanically connecting/disconnecting a link.

Other experiments have been conducted in order to measure the performance of CANcentrate. Two important results have been obtained. First, the average of the extra delay introduced by the hub is 155 ns (in particular, the total extra delay introduced by the two transceivers of the Input/Output Module is 120 ns). The internal part of the hub was also built with 8 and 16 ports. The measurements indicate that the extra delay introduced by the internal part of the hub does not visibly depend on the number of ports it is provided with. Additionally, several cables of different lengths as well as different bit rates were used in order to measure the performance of the network with respect to the star diameter and the bit rate. Due to implementation limitations, the maximum used bit rate was 690kbit/sec. At this bit rate, normal communication was achieved with a star diameter of 70 meters.

With such diameter, the turnaround time was about 1180ns, which includes two times: the propagation delay through the cable (70m*4.5ns/m=315ns), the delay of four transceivers (4*60ns=240ns), and the delay of the internal part of the hub (35ns). In order to work at 690kbit/sec, the bit time was divided into 8 time quanta of 181ns and the sample point was approximately located at 1267ns, which is far enough from the beginning of the bit time to take into account the turnaround time. With 80 meters the communication capabilities were disrupted. Notice that the maximum length of a bus operating at the same bit rate would be around 100 meters.

VI. Conclusions

The use of CAN in safety-critical applications has been controversial due to a few factors, such as its bus topology. Communication systems relying on simplex or replicated bus topologies suffer from several impediments to enforce error containment, even if they are used together with bus-guardians. In contrast, in star topologies the hub can play a key role to diagnose and passivate faults, thereby reducing the number of components whose failure can cause a severe failure of communication, to a unique single point of failure, i.e. the hub.

In this paper we proposed a new active star topology, called CANcentrate, that is compatible with commercial off-the-shelf CAN components and that can be used with any CAN-based protocol (e.g. TTCAN [9], FTT-CAN [17], Timely CAN [18], etc).

We explained the architecture and the fault-treatment mechanisms of the central device of CANcentrate, a hub, which can be built using off-the-shelf FPGA technology. Moreover, we described the implementation and test of a first prototype of CANcentrate that we have built.

In general, CANcentrate is a further step towards improving dependability of CAN networks.

References

[1] S. Cavalieri, "Meeting Real-Time Constraints in CAN," *IEEE Transactions on Industrial Informatics*, pp. 124–135, May 2005.

[2] T. Nolte, M. Nolin, and H. A. Hansson, "Real-Time Server-Based Communication with CAN," *IEEE Transactions on Industrial Informatics*, pp. 192–201, August 2005.

[3] M. Barranco, L.Almeida, and J. Proenza, "ReCANcentrate: A replicated star topology for CAN networks," *ETFA 2005. 10^{th} IEEE International Conference on Emerging Technologies and Factory Automation, Catania, Italy*, 2005.

[4] ISO, "ISO11898. Road vehicles - Interchange of digital information - Controller area network (CAN) for high-speed communication," 1993.

[5] I. Broster and A. Burns, "An Analyzable Bus-Guardian for Event-Triggered Communication," in *Proceedings of the 24th Real-time Systems Symposium (RTSS)*. Cancun, Mexico: IEEE, Dec 2003, pp. 410–419.

[6] J. Rufino, P. Veríssimo, and G. Arroz, "A Columbus' Egg Idea for CAN Media Redundancy," *FTCS-29. The 29th International Symposium on Fault-Tolerant Computing, Winconsin, USA*, June 1999.

[7] J. Rushby, "A Comparison of Bus Architectures for Safety-Critical Embedded Systems," SRI International, Menlo Park, California," Contractor Report, 2003.

[8] L.-B. Fredriksson, "CAN for critical embedded automotive networks," *IEEE Micro, Special Issue on Critical Embedded Automotive Networks*, vol. 22, no. 4, pp. 28–35, July-August 2002.

[9] H. Kopetz, "Time-Triggered Protocols for Safety-Critical Applications," Presentation, Vienna University Of Technology, TU Wien, Karlsplatz 13, 1040 Vienna, Austria, March 2003.

[10] M. Rucks, "Optical layer for CAN," *1st International CAN Conference*, November 1994.

[11] CiA, "CAN physical layer," CAN in Automation (CiA), Am Weichselgarten 26, Tech. Rep. [Online]. Available: headquarters@can-cia.de

[12] IXXAT, "Innovative products for industrial and automotive communication systems," 2005. [Online]. Available: http://www.ixxat.de/index.php

[13] G. Cena, L. Durante, and A. Valenzano, *A new CAN-like field network based on a star topology*, Polytechnic Institute Torino Std. 23, July 2001.

[14] M. Barranco, G. Rodríguez-Navas, J. Proenza, and L. Almeida, "CANcentrate: An active star topology for CAN networks," *WFCS'04. IEEE Workshop on Factory Communication Systems, Vienna, Austria*, 2004.

[15] M. Barranco, J. Proenza, G. Rodríguez-Navas, and L. Almeida, "A CAN hub with Improved Error Detection and Isolation," *10th International CAN Conference*, March 2005.

[16] P. Fonseca, A. Santos, and J. Fonseca, "A dynamically reconfigurable CAN system," *Proceedings of the 5th International CAN Conference*, 1998.

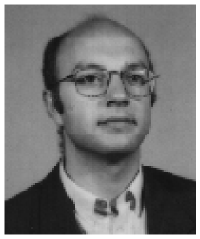
[17] L. Almeida, P. Pedreiras, and J. A. Fonseca, "The FTT-CAN Protocol: Why and How," in *IEEE Transactions on Industrial Electronics - special issue on Factory Communication Systems*, vol. 49, no. 6, December 2002.

[18] 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 Conference on Real-time Systems (ECRTS)*. IEEE, 2001, pp. 95–102.

Manuel Barranco received the first degree in Informatics Engineering from the University of the Balearic Islands (UIB), Palma de Mallorca, Spain, in 2003, and he is currently a Ph.D. candidate in the Department of Mathematics and Informatics at the UIB. He is also a member of the System, Robotics and Vision (SRV) research group at this university. His research interests include dependable and real-time systems, fault-tolerant distributed embedded systems, event-triggered and time-triggered communication sytems, and field-bus networks such as the Controller Area Network (CAN).

Julián Proenza received the first degree in Physics from the University of the Balearic Islands (UIB), Palma de Mallorca, Spain, in 1989, and he is currently a Ph.D. candidate in the Department of Mathematics and Informatics at the UIB. He is currently a Lecturer in the Department of Mathematics and Informatics at the UIB. His research interests include dependable and real-time systems, fault-tolerant distributed systems, clock synchronization and field-bus networks such as CAN (Controller Area Network).

Guillermo Rodríguez-Navas received the first degree in Telecommunication Engineering from the University of Vigo, Spain, in 2002. He is currently doing a PhD in Computer Science at the University of the Balearic Islands, Spain. He is also a member of the System, Robotics and Vision (SRV) research group at this university. His research is focused on dependable and real-time distributed embedded systems. In particular, he has addressed various issues related to the Controller Area Network (CAN) field bus, such as fault tolerance, clock synchronization and response time analysis.



Luís Almeida graduated in Electronics and Telecommunications Engineering at the University of Aveiro in 1988 and received a PhD in Electrical Engineering in 1999 from same University. He is an Assistant Professor at the Electronics and Telecommunications Dep of the University of Aveiro, Portugal, and a senior researcher at the IEETA research unit of the same university. Formerly he was a design engineer in a company producing digital telecommunications equipment. His research interests lie in the fields of real- time networks for distributed industrial/embedded systems and control architectures for mobile robots. He is a Senior member of the IEEE Computer Society.