Parameterizable CAN Switch Implementation Using FPGA

João Faria, Samuel Madail, Arnaldo Oliveira

***Abstract* – The Controller Area Network is a well known fieldbus commonly used in many distributed control systems. However, the original bus topology of CAN limits its usage in safety-critical and fault tolerant applications. In order to improve its performance and allow its deployment in safety-critical applications, several CAN hubs and switches have been created, permitting the adoption of star-based topologies. This paper presents the architecture, FPGA implementation and test of a parameterizable CAN 2.0B switch based on a synthesizable CAN intellectual property core developed at DETI-UA. The switch consists of multiple instantiated CAN controllers, the pool of message buffers and the logic required to forward messages between ports. All switch core components are implemented in FPGA logic cells, except the CAN physical layer transceivers.**

***Resumo* – O barramento CAN (Controller Area Network) é amplamente utilizado em sistemas envolvendo controlo distribuído. Apresenta no entanto algumas reservas inerentes à sua topologia e que limitam a sua utilização em sistemas de segurança críticos e tolerantes a falhas. Para melhorar o desempenho têm sido desenvolvidos vários hubs e switches, proporcionando a adopção de topologias baseadas em estrela. Este artigo apresenta a arquitectura, implementação e teste de um switch CAN 2.0B parametrizável e baseado num núcleo de propriedade intelectual sintetizável de um controlador desenvolvido no DETI-UA. O switch consiste na instanciação de múltiplos controladores CAN, blocos de memória para armazenamento das mensagens e toda a lógica para reencaminhamento das mesmas entre as portas do switch. Todos os componentes foram implementados em FPGA excepto os transceivers CAN responsáveis pela camada física.**

***Keywords* – Controller Area Network (CAN), Field buses, Fault tolerant communication system, Switch, Real-Time, FPGA.**

***Palavras chave* – Controller Area Network (CAN), Barramentos de campo, Sistemas de comunicação tolerantes a falhas, Switch, Tempo-Real, FPGA.**

I. INTRODUCTION

The Control Area Network (CAN) protocol [1] is a field bus widely used due to its high reliability, acceptable real-time performance and low cost. These characteristics make CAN a good choice for many distributed embedded control systems, such as factory automation or in-vehicle communication. CAN is based on a bus topology and employs a Carrier Sensing Multiple Access/Collision Avoidance (CSMA/CA) medium access control mechanism consisting of a dominant/recessive arbitration bit (see figure 1).

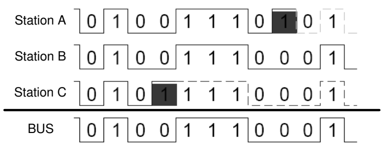


Fig. 1 - CAN recessive/dominant arbitration bit mechanism.

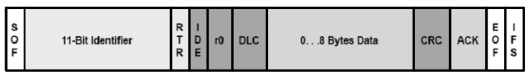


Fig. 2 - CAN frame structure.

During arbitration, each transmitter node or station compares the bit it is trying to transmit with the bit present in the bus. If it senses a dominant bit while transmitting a recessive bit, it means that another transmitter is sending a higher priority message. The node that loss arbitration stops the transmission and delays the sending of its message until the next arbitration round, which guarantees that no information is lost.

In a CAN-based distributed system the information is exchanged in fixed format messages of limited length (see figure 2). Each message has an identifier which defines a static message priority used during bus arbitration. The identifier is also used to inform the remaining network nodes about message's content. CAN is based on the consumer-producer paradigm, which means that the message identifier carries information about its contents, not the identification or address of the source or destination nodes. Therefore, all nodes in a CAN network decide by themselves about the reception (or discard) of a message. It means that any group of nodes can receive and simultaneously act upon the same message. Although each message identifier can only be produced by one node in the network.

In a bus-based communication system like CAN, the network nodes or stations have direct electrical connections to the shared medium, which is the main weakness and impairment form the dependability point of view. A failure in any node of the network can prevent correct communication between any other nodes. In [2] some possible malfunctions of a CAN-based communication system are summarized (e.g. a failure in the medium access control of a node can produce dominant bits, which block the communication channel and none of the nodes can communicate). Because a unique bus is shared by all communication paths between every subset of nodes, a failure in just one point causes a breach of many communication paths. Another failure can occur when a partition takes place in the local connection of a node. In this case, the node gets isolated and there is no influence in the communication between the others. Due to the several specific dependability problems presented and caused by the bus topology, a topology, called CANCentrate, based on a active hub has been proposed [3]. It has mechanisms that reduce the number of severe points of failure to a unique single point of failure, namely the hub. However, with a switch there are many other possibilities, such as parallel communication paths and interconnection of CAN networks operating at different bit rates.

This paper is organized as follows: Section 2 describes the motivation of this work. Section 3 summarizes some related works and publications. Section 4 describes the architecture of the system. Section 5 presents the modeling, implementation and prototyping approach. The results of evaluation and test are shown in section 6. Finally, section 7 presents the conclusions and some possible directions for future work and improvements.

II. MOTIVATION AND OBJECTIVES

Nowadays distributed embedded systems that require real-time performance need a network capable of deterministic access delay. In this context, CAN (Controller Area Network) became well-known in the last years due to its low price, electrical robustness and priority-based access control. However, the main disadvantage of any protocol using a bus topology is that a single network segment is shared between multiple components with direct electrical connections to each other and without proper error confinement. As a consequence, it presents dependability limitations that are inherent to its bus topology. In particular CAN lacks of the tailored mechanisms to avoid that errors generated by a single fault jeopardize the communication capabilities of many nodes, possibly causing a general failure of the system.

In order to solve some of those problems, an active hub implementing the CAN fault confinement mechanisms can be a good choice. However, if we want to connect networks with different speed rates (act as a rate converter), forward or send selectively CAN messages only in certain ports (act as a multiplexer or demultiplexer), select the direction of message propagation in subsystems (like a CAN diode) or even prevent the propagation of error messages across the entire network, it becomes necessary to use a CAN switch. In fact, the use of switch in a CAN network brings more flexibility, reliability and robustness and a set of features not provided by a hub. Moreover, a hub shares the total bandwidth among all users, while a switch provides a dedicated line at full bandwidth between every two devices transmitting to each other. It means that the usage of a switch is much more efficient in terms of bandwidth than the usage of an hub. In the 1990s, switches were much more costly than hubs, and devices were carefully evaluated based on the traffic requirements. By the turn of the century, switches became much less expensive, and the popularity of hubs began to wane.

A. Switch functionalities

As described in [4], a CAN switch can have the following functionalities (shown in figure 3):

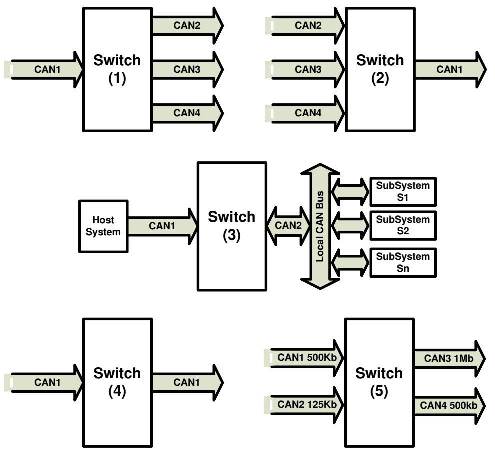


Fig. 3 - Examples of a CAN switch functionalities.

1. **Demultiplexer** - The data from one CAN channel can be demultiplexed to two or more channels. The data in each output channel can be a filtered subset of the data available in the input channel.
2. **Multiplexer** - The data from two or more CAN channels can be multiplexed into one CAN channel.
3. **CAN Data Diode** - The input CAN data bus can be isolated from the output CAN bus. This allows for several systems to hang off of the output CAN bus and communicate to each other without having data broadcast on the input CAN bus. In the example (3) in figure 3, the Host System CAN data is received on CAN1 and broadcast out on CAN2. The subsystems are tied together on the common CAN2 bus. The subsystems can communicate between themselves and receive data from the Host System without broadcasting any information on the Host System CAN bus.
4. **Echo** - The data received on the input CAN bus can be echoed back onto the same CAN bus.
5. **Rate Converter** - Data can be received on the input CAN channel at one baud rate and broadcasted on the output CAN channel at a different baud rate. In the example (5) presented in figure 3 CAN1 data is received at 500Kb/s and broadcasted on CAN2 at 1Mb/s. CAN3 is received at 16.5Kb/s and broadcasted on CAN4 at 500Kb/s.

A switch can also be a great help concerning the growth in length of wiring that leads to excessive bus loading. Some relief can be achieved by subdividing the system into multiple CAN networks. The purpose of a switch (like a gateway) is to transfer to the other CAN network only those messages that are of interest for that network. For example, a switch is ideal for software developers of electronic control units in motor vehicle applications. It allows to perform complex handling of multiple CAN networks using a standard implementation that is easier to integrate. Other advantage is that it permits data exchange between the networks in two different routing levels: signal routing and message routing. Furthermore, there are several ways to implement a CAN switch. It can be implemented in software running on processor and compiled from high-level languages (e.g. C/C++) allowing fast and easy changes of implementation configurations. It can be programmed in assembly language compiled for a RISC processor to achieve a low latency between reception and transmission of a CAN message. On the other hand, it can be implemented fully in hardware based on a ASIC, which allows achieving the highest performance and the lowest cost for high volume markets. Finally, it can also be executed in FPGA with the core logic developed using VHDL, which combines the advantages of both traditional hardware and software design techniques. FPGAs have the ability to deliver the necessary speed and parallelism of ASICs, while maintaining the re-configurability and flexibility of software. So FPGAs could be a great way to implement a CAN switch due to low latency between reception and transmission of a CAN message. Its use with the VHDL language and automatic synthesis tools also allows fast design cycles and great flexibility, such as in the number of ports or message memory dimensions.

III. RELATED WORK

The CAN protocol is implemented in several applications and its use is widespread in distributed embedded systems due to its electrical robustness, low price and deterministic access delay. Therefore, there is a great effort to develop ways to solve the limitations imposed by the network topology and make it more robust for real-time and safety critical applications. Besides, there is still a high interest in finding new solutions based on the well studied properties of CAN [5], [6].

In [2], [3], [7] it is reported the development and results of a CAN hub created to overcome the inability of preventing global communication failures in CAN bus based system (when a fault in some component of bus occurs). The authors proposed a design of hub-based active star topology called CANcentrate. Their hub is an active hub that prevents error propagation from any of its ports to the others and it is fully compatible with existing CAN controllers. From the theoretical analysis point of view, they model both the bus-based and the active star CAN communication systems using stochastic Petri Nets, and then evaluate which is their probability of not suffering a severe failure. The first results show that, for a given number of nodes, the CANcentrate is clearly more resistant to faults than a CAN bus throughout the time. Moreover, even when considering a CANcentrate network with more nodes than a CAN bus, CANcentrate is better than CAN during some thousands of hours [2], [3], [7].

For improved fault tolerance, the same research group that developed and evaluated CANcentrate is also implementing ReCANcentrate that is constituted by two or more hubs working in parallel [8], [9]. Each pair of hubs is coupled by means of two ore more special links called interlinks. Within a fraction of the bit time, each hub sends to the other hubs its resultant signal, i.e., the signal that results from coupling the contributions of its own nodes and, at the same time, it couples this signal with the one received from the other hubs and broadcasts the result to its own nodes. The ReCANcentrate hub includes the same error detection mechanisms as in CANcentrate and additionally it is able to detect and isolate a interlink or a given hub that is stuck-at or bit-flipping. Moreover, a node is able to detect when a given hub is faulty using the error detection mechanisms of any commercial CAN controller and, then, it can stop using a faulty hub. Therefore, nodes can still communicate as long as there is a non-faulty hub [8], [9]. Both CANcentrate and ReCANcentrate active hub topologies need only an extra transceiver at a node to connect it to its hub port, i.e., one transceiver will connect it to the uplink and the other to the downlink.

There is already an Embedded Software Component for CAN-CAN Routing in vehicles designated by CANbedded gateway [10]. The gateway is ideal for software developers of electronic control units in motor vehicle applications. It allows them to perform complex handling of multiple CAN networks using a standard implementation that is easier to integrate. The gateway component, supplied in the form of C source code, permits data exchange between the networks on two different routing levels: Signal routing and Message routing (with or without data queuing). The required routing algorithm and the signals and messages to be routed are automatically selected based on database attributes in the communication matrix. The specifications of motor vehicle OEMs are considered in the process. It is also possible to perform manual configuration on the signal or message level in the generation tool [10].

There are also some commercial tools such as a routing gateway called CAN Matrix [4]. It is a multi-configurable CAN gateway with several base and routing configurations that can be used. The system is compliant with CAN2.0b specification handling both standard 11-bit and extended 29-bit header (J1939) based on High Speed Dual Wire diferencial electrical physical layer. The tool supports up to five independent input CAN channels and up to five independent output channels. The CAN Matrix provides 5 optically isolated digital outputs that can be triggered on the reception of a specific CAN message on any one of the five channels. Two additional RS232 serial ports can be used for high speed serial to CAN conversion. Each serial data byte is converted into a CAN message with a data length of 2 bytes. The tool is based on a RISC processor that, accordingly to its developers, minimizes the latency between reception and transmission of a CAN message. The RISC processor is responsible only for receiving CAN messages and then transmitting them on the appropriate output CAN channel. Another processor handles all of the programming and housekeeping.

Besides the claims of CAN Matrix developers regarding short latency, we believe that better results can be achieved with an FPGA-based hardware implementation, that combines the advantages of an hardware implementation with the flexibility of a software implementation. This makes us to believe that this work has an extreme interest owing to the great advantages of switch CAN described in the previous section.

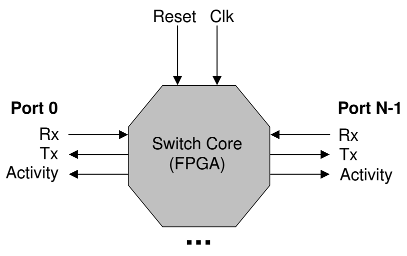


Fig. 4 - Interface of the CAN Switch.

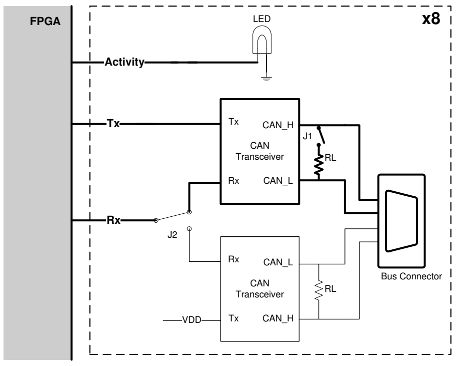


Fig. 5 - Structure of a CAN Switch port.

IV. ARCHITECTURE

In this section the architecture of the developed CAN Switch is presented. The CAN Switch core, implementing the message storing and forwarding functionalities, was implemented in FPGA. It was modeled in VHDL allowing the parametrization of the number of switch ports and the number of message buffers available. Figure 4 represents the interface of the CAN Switch, consisting of the signals used to connect each port to a CAN transceiver (*canRx* and *canTx*), the signals used to indicate the activity in each bus, the synchronization input signal connected to an 48MHz oscillator and a reset pin.

The switch was implemented using the Celoxica RC-10 FPGA development kit. However, this board only provides the FPGA and some auxiliary components. The CAN physical layer transceivers and connectors required to connect the switch to the CAN busses are provided in a extension daughter printed circuit board connected to the RC-10 through the I/O expansion headers.

Figure 5 depicts how the CAN Switch ports are connected to the FPGA. Each port has one led used to indicate the bus activity, one CAN bidirectional transceiver that provides the standard half-duplex transmit/receive capabilities between the bus and the CAN controller and one jumper that permits the insertion or removal of the termination resistor between the CAN Hi and CAN Lo diferencial lines. To make the board useful to other CAN network topologies and compatible with some expected future developments, such as CANcentrate which was mentioned above and employees an active hub with unidirectional busses, an additional transceiver was added. It offers the possibility of simultaneously transmission and reception [2]. As defined by some CAN physical layers, the bus connectors used were DB9 and the second transceiver is connected to unused pins in the standard CAN connector. The developed board has eight ports and two socket for custom oscillators.

The internal structure of CAN Switch is shown in Figure 6 and is described next. A CAN protocol controller is required for each switch port. It is implemented by the CLAN intelectual property synthesizable core. CLAN core is a CAN 2.0B controller developed at Electronics, Telecommunications and Informatics Department of University of Aveiro, was the CAN Controller used and it is described in detail in [11]. It implements the link between physical interface and the higher layers and is responsible for sending and receiving CAN messages. It is able to work with CAN 2.0A and with CAN2.0B specifications. In this application it is programmed to operate at 1Mbit/seg, but it can be easily changed to operate with others speed rates as shown in [11]. The remaining of the switch consists of additional logic and buffers to store and forward messages correctly. Switch works in a time division multiplex way in which ports are served cyclically in a static order. As the frequency is not high enough to analyze all ports during a CAN bit time, additional logic is needed to register some signals generated by CLAN core. When a certain port is attended the presence of a received message is analyzed. If a message is received, it is stored in memory into a position determined and assigned by the memory management unit. Simultaneously, based on the ID of the received message, the forwarding unit determines the destination ports in which the message should be transmitted. Each port has an associated memory in which information about messages to be sent on that port is stored. This memory is a First-In-First-Out memory and data stored on it corresponds to the position in memory of the messages to be sent. As in the reception, during transmission each port is attended one by one. If the FIFO associated with the port being attended is not empty and the corresponding bus is free, the position of the next message to be transmitted on that port is read and then message is transmitted. When a message is transmitted the memory management unit is responsible for refresh the memory containing the output ports of each message. When a message is transmitted to all destination ports the memory buffer is released and available to receive a new message. The position to store the received message is obtained using a priority encoder which determines the first memory address free. In this first prototype the forwarding unit is implemented based on a static table. This table is used to determine the ports to which each message must be sent to and the search can be done in four clock cycles. The memory to store messages has capacity to store up to sixteen different messages. The described architecture is represented in figure 6.

V. MODELING, IMPLEMENTATION AND PROTOTYPING

The CAN Switch was modeled using VHDL (Very High Speed Integrated Circuit Hardware Description Language). The switch is parametrizable, allowing the number of ports and memory sizes to be defined during synthesis. In addition to the files created to model the switch, the CLAN IP core files need to be included [11], namely: the

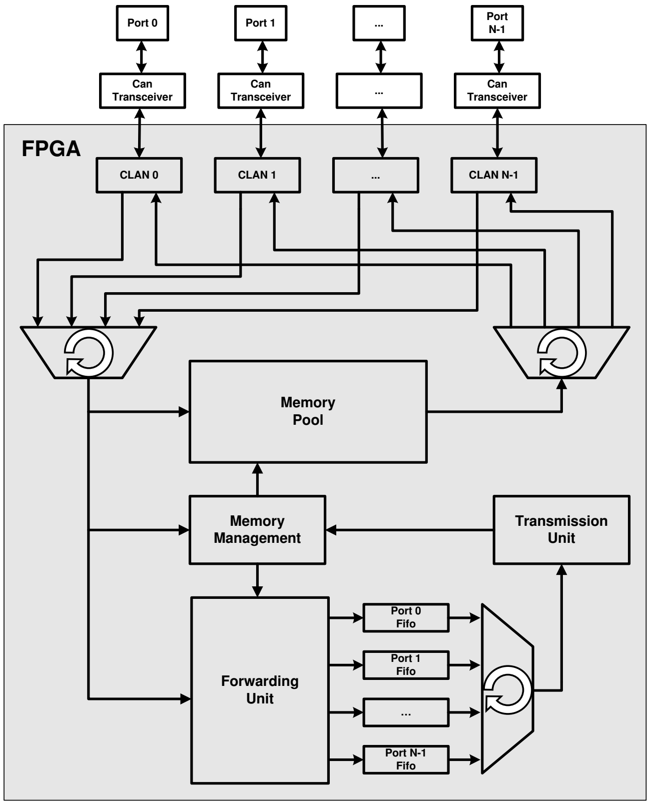
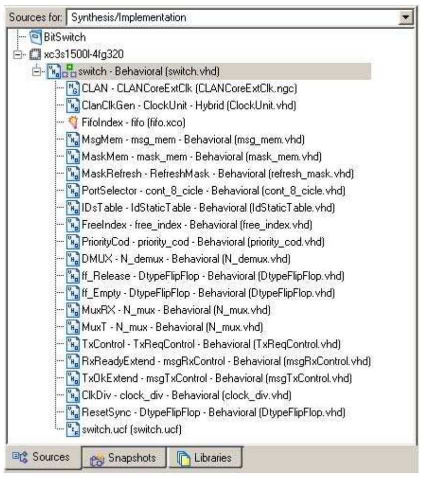


Fig. 6 - CAN Switch internal block diagram.



```
Sources for: Synthesis/Implementation
BitSwitch
xc3s1500l-4fg320
  switch - Behavioral (switch.vhd)
    CLAN - CLANCoreExtClk (CLANCoreExtClk.ngc)
    ClanClkGen - ClockUnit - Hybrid (ClockUnit.vhd)
    FifoIndex - fifo (fifo.xco)
    MsgMem - msg_mem - Behavioral (msg_mem.vhd)
    MaskMem - mask_mem - Behavioral (mask_mem.vhd)
    MaskRefresh - RefreshMask - Behavioral (refresh_mask.vhd)
    PortSelector - cont_8_cicle - Behavioral (cont_8_cicle.vhd)
    IDsTable - IdStaticTable - Behavioral (IdStaticTable.vhd)
    FreeIndex - free_index - Behavioral (free_index.vhd)
    PriorityCod - priority_cod - Behavioral (priority_cod.vhd)
    DMUX - N_demux - Behavioral (N_demux.vhd)
    ff_Release - DtypeFlipFlop - Behavioral (DtypeFlipFlop.vhd)
    ff_Empty - DtypeFlipFlop - Behavioral (DtypeFlipFlop.vhd)
    MuxRX - N_mux - Behavioral (N_mux.vhd)
    MuxT - N_mux - Behavioral (N_mux.vhd)
    TxControl - TxReqControl - Behavioral (TxReqControl.vhd)
    RxReadyExtend - msgRxControl - Behavioral (msgRxControl.vhd)
    TxOkExtend - msgTxControl - Behavioral (msgTxControl.vhd)
    ClkDiv - clock_div - Behavioral (clock_div.vhd)
    ResetSync - DtypeFlipFlop - Behavioral (DtypeFlipFlop.vhd)
    switch.ucf (switch.ucf)
```

Fig. 7 - Hierarchy of the CAN Switch project.

CLANCore.ngc file containing the synthesized netlist, the *CAN.vhd* file containing a package with generic CAN definitions and the *CLANPublic.vhd* file containing a package with CLAN project specific definitions. The entire project hierarchy with all files needed for synthesis and implementation is shown in figure 7

The final synthesis report can be seen in figure 8. The CAN Switch was synthesized and implemented on a Xilinx XC3S1500L Spartan-3L FPGA. It can operate up to 52Mhz and occupies about 40% of the available slices (logic cells).

The switch was prototyped using a Celoxica RC-10 development board extended with the CAN transceivers board referred above. Figure 9 shows the developed proptotype.

Fig. 8 - Xilinx ISE synthesis report for the CAN Switch.

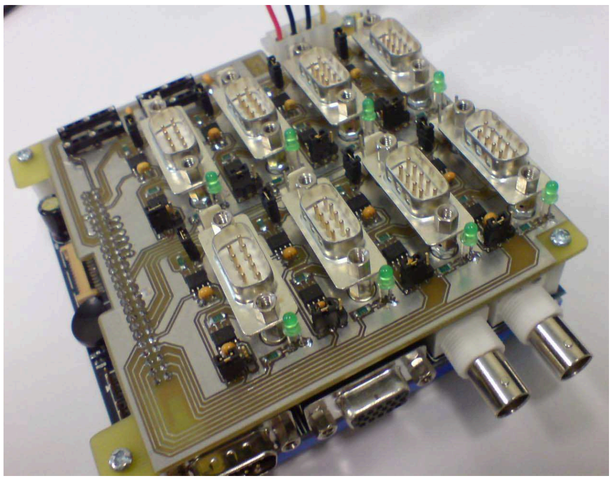


Fig. 9 - CAN Switch prototyping board.

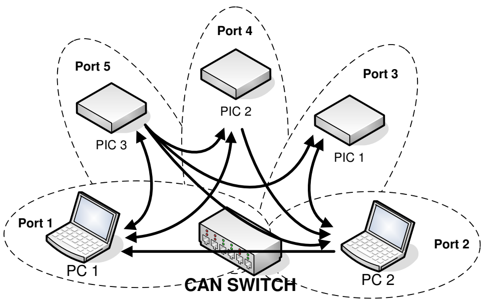


Fig. 10 - CAN Switch evaluation setup.

VI. TEST AND EVALUATION

To test the CAN Switch, the network shown in figure 10 was created. It consists of two personal computers connected to the switch using two PCAN-USB devices and three CAN PIC microcontrollers. The content of the forwarding table used is presented on the table I, where the outgoing ports for each message received are represented.

Network nodes configuration can be seen in the second and third columns of the table II. They represent the ID of messages produced in each node and their period, respectively. In fourth and fifth columns are indicated the messages received by each node and respective period.

After observing the results, it is possible to conclude that CAN Switch is working properly. The internal processing latency measured from reception to transmission was $12\mu s$.

Memory Initialization							
ID	1	2	3	4	5	10	11
Outgoing Ports	1	2	3	4	5	1,2	1,2,3,4

TABLE I
EXAMPLE OF A FORWARDING TABLE.

Port	Produced Messages	Period	Consumed Messages	Period
1	4, 5	200 ms	1	200 ms
			10, 11	100 ms
2	1, 3	200 ms	2, 10, 11	100 ms
3	2	100 ms	3	200 ms
			11	100 ms
4	10	100 ms	4	200 ms
			11	100 ms
5	11	100 ms	5	200 ms

TABLE II
EVALUATION RESULTS.

This value was obtained by sending a periodic message with period $1ms$ to port 2 and receiving it in port 1. This time is the interval between the end of the message sent to CAN switch and the start of message in the outgoing port. Considering the architecture of the switch and due to the loop to attend all ports, the presented situation corresponds to the worst case.

VII. CONCLUSION AND FUTURE WORK

A CAN Switch prototype developed for educational and research purposes was presented in this paper. Although it is working correctly, it is important to refer that this is the first implementation and many aspects must be improved. Some limitations of the architecture and possible solutions need to be addressed. First of all, the cyclic or TDMA processing of all ports means that the processing latency depends on the particular ports used. To improve that, instead of multiplexing all ports over time, a parallel processing scheme should be implemented to decrease the latency. The storage capacity needs also to be increased because resources can easily be exceeded if it is received a burst of messages to be forwarded to the same port, because it has capacity to store only sixteen messages. A better searching algorithm is also needed. Currently, searching in the forwarding table is done by looking sequentially for all entries until the desired ID is found. To speedup this process, this table should be replaced by a content addressable memory (CAM), which is expected to improve the performance of the searching algorithm. Finally, message forwarding is based on a static table. Although it would be interesting to develop an automatic mechanism that allows nodes to dynamically register themselves in the switch as consumers of messages with particular IDs. One possible solution for this is the usage of the CAN Remote Transmission Request (RTR) frames. This type of frames allows any node request the transmission of a particular data message. It can also be used to inform the switch that it wants to receive messages with the ID contained in RTR message. The dynamic update of the forwarding table based on aging mechanisms also permits the insertion or removal of nodes without explicitly changing the forwarding table. If a message is not received during a certain period it could mean that the respective node was removed and the corresponding entry of the forwarding table can be removed. This contributes to a more efficient use of the several switch memories.

REFERENCES

[1] Robert Bosch GmbH, “CAN specification version 2.0”, http://www.can.bosch.com, Sept. 1991.

[2] Manuel Barranco, Julián Proenza, Guillermo Rodríguez Navas, and Luis Almeida, “An active star topology for improving fault confinement in CAN networks”, *IEEE Transactions on industrial informatics*, vol. 2, pp. 78–85, 2006.

[3] Manuel Barranco, Guillermo Rodriguez-Navas, Julian Proenza, and Luis Miguel Pinho de Almeida, “CANcentrate: an active star topology for can networks”, in *IEEE International Workshop on Factory Communication Systems*, 2004, pp. 219–228.

[4] CANMATRIX, “CAN bus routing gateway data sheet”, http://www.eritools.com/CANMATRIX.pdf, 2006.

[5] S. Cavalieri, “Meeting real-time constraints in CAN”, *IEEE Transactions on industrial informatics*, vol. 1, pp. 124–135, 2005.

[6] T. Nolte, M. Nolin, and H. A. Hansson, “Real-time server-based communication with CAN”, *IEEE Transactions on industrial informatics*, vol. 1, pp. 192–201, 2005.

[7] M. Barranco, J. Proenza, and L. Almeida, “First results of the assessment of the improvement of error containment achieved by cancentrate”, *IEEE International Workshop on Factory Communication Systems*, pp. 75–78, 2006.

[8] M. Barranco, L. Almeida, and J. Proenza, “ReCANcentrate: A replicated star topology for CAN networks”, *ETFA 2005. 10th IEEE International Conference on Emerging Technologies and Factory Automation, Catania, Italy*, vol. 2, pp. 469–476, 2005.

[9] M. Barranco, L. Almeida, and J. Proenza, “Experimental assessment of ReCANcentrate, a replicated star topology for CAN”, *SAE 2006 World Congress, Detroit, Michigan, USA*, 2006.

[10] Vector Informatik GmbH, “CANbedded Gateway”, www.vector-informatik.com.

[11] A. S. R. Oliveira, N. L. Arqueiro, and P. N. Fonseca, “CLAN - a technology independent synthesizable can controller”, *CAN in Automation*, pp. 1–8, 2005.

[12] M. Barranco, J. Proenza, G. Rodriguez-Navas, and L. Almeida, “A can hub with improved error detection and isolation”, in *ICC 2005, 10th Int. CAN Conference*, 2005.

[13] Manuel Barranco, Guillermo Rodriguez-Navas, Julian Proenza, and Luis Miguel Pinho de Almeida, “Pushing error containment and fault tolerance in CAN by means of star topologies: CANcentrate and ReCANcentrate”, http://dmi.uib.es/ mbarranco/srvlsestars/description.htm.