A VHDL CAN MODULE FOR SMART SENSORS

José E. O. Reges, Edval J. P. Santos

Laboratory for Devices and Nanostructures / Universidade Federal de Pernambuco,
Rua Acad. Hélio Ramos, s/n, Várzea, 50740-530 Recife, PE, Brasil
email: jose_edenilson@yahoo.com.br, edval@ee.ufpe.br

ABSTRACT

Communication modules are required for smart sensors interface with the sensor network. In this paper, a VHDL (*VHSIC Hardware Description Language*) implementation of a CAN (*Controller Area Network*) interface for smart sensors is presented. The scope of this paper is the Medium Access Control (MAC) sublayer. Therefore, it deals with the transfer protocol, control of frames, arbitration, error checking and error signaling. In accordance with the CAN protocol (versions 2.0A and 2.0B), the interface can be divided into blocks to perform various communication tasks. In the implementation presented in this work, each block is a VHDL project entity described in the behavioral style. The performance of each entity is analyzed separately. Next, all entities are interconnected in the structural style. The final description has been synthesized into a *Xilinx® Spartan-II XC2S50* FPGA. Finally, a comparison between this implementation and the *HurriCANe*, a freely available core, is performed.

1. INTRODUCTION

CAN (*Controller Area Network*) is a serial bus interface, which was originally developed in the early 1980's for automotive applications [1]. This protocol efficiently supports distributed real-time control with a very high level of security [2]. Currently, CAN is also used in other applications, such as: maritime electronics; aircraft and aerospace electronics; medical equipments and devices; automated control systems [1].

CAN has the following properties: prioritization of messages; guarantee of latency times; configuration flexibility; multicast reception with time synchronization; system wide data consistency; multi-master; error detection and signaling; automatic retransmission of corrupted messages as soon as the bus is idle again; distinction between temporary errors and permanent failures of nodes and autonomous switching off of defective nodes [2]. Such characteristics are very attractive to make CAN a suitable communication protocol for smart sensors networks wired and wireless.

The authors acknowledge the support of CNPq, FINEP, PETROBRAS and XILINX.

Some microcontroller and FPGA (*Field Programmable Gate Arrays*) manufacturers offer a broad range of *off-the shelf* solutions directed to CAN, such as: microcontrollers with CAN controller, stand-alone CAN controllers and IP cores. Such as: the *Philips®* P87C591 Microcontroller with CAN controller; the *Microchip®* MCP2510 stand-alone CAN controller; the C_CAN, a VHDL CAN module developed by *Bosch®*. However, these solutions have disadvantages regarding performance and cost.

Microcontrollers with integrated CAN interface suffer a performance penalty, as the microcontroller is responsible for message transmission and reception, besides reading inputs and driving outputs. This is a critical factor in industrial networks, where latency is an issue. If a stand-alone CAN controller is used, there is a cost penalty as an extra IC is required, increasing the cost of the system. Finally, IP cores developed by FPGA manufacturers and independent designers are generally not free. All these factors evidence the necessity of development of a CAN module for smart sensors.

Using the ISO/OSI reference model, the CAN protocol is subdivided into different layers: the Data Link Layer (DLL) and the Physical Layer. The DLL is subdivided into two sublayers: the Logical Link Control (LLC) sublayer and the Medium Access Control (MAC) sublayer [2].

The scope of this paper is the Medium Access Control (MAC) sublayer. Therefore, the focus in this paper is the transfer protocol, control of frames, arbitration, error checking and error signaling.

To synthesize this design, the *Xilinx® Spartan-II XC2S50* FPGA has been selected, as the development platform. Conditioning circuits and transducers are custom designed with *Mentor®* tools. The VHDL description will be integrated with the custom designed cells using *Mentor®* tools. The paper is divided into four sections, this introduction is the first. Next, the description of the project entities is presented. Third, the simulation and synthesis results are discussed. Finally, the conclusions.

2. DESCRIPTION OF THE PROJECT ENTITIES

The CAN module block diagram is presented in Figure 1. This module is composed by four basic entities: CAN TX, CAN RX, STUFF HANDLER and CRC.

978-1-4244-1992-0/08/$25.00 ©2008 IEEE.

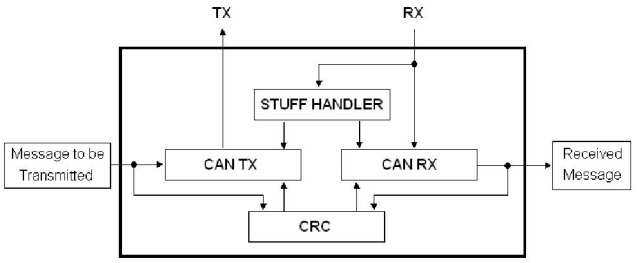

Fig. 1. CAN module block diagram.

CAN TX and CAN RX are the two most important entities, performing the transmission and reception tasks, respectively. STUFF HANDLER and CRC are auxiliary entities for network failure detection.

To implement the CAN protocol in VHDL, the "divide to conquer" approach is applied. The basic entities are described using the behavioral style to determine the performance bottlenecks. After completing the evaluation, the entities are linked to a main entity using the structural style. The performance of each entity is analyzed separately.

2.1 CAN TX entity

The CAN TX entity is responsible for the transmission of messages, in accordance with the CAN protocol. CAN TX verifies if there is a message to be transmitted and if the bus is idle. When a transmission is started, this entity verifies the bus state, comparing the TX bit and RX bit.

When TX bit is recessive (logical level 1) and RX bit is dominant (logical level 0), the entity assumes that the unit lost the bus arbitration and must withdraw without send any more bits. When a dominant bit is sent and a recessive bit is received, the entity assumes an error has occurred and stops transmission.

When the STUFF HANDLER entity informs the need of a *stuff bit*, CAN TX automatically sends a complementary bit to the last transmitted bit.

After sending the Data Field, CAN TX requests that the CRC entity send the CRC Sequence.

2.2 CAN RX entity

This entity receives and stores messages in the CAN format. Moreover, it also receives and stores the Message Identifier. This field, besides defining priorities, is used by a CAN node to filter the messages which are addressed to it.

When the STUFF HANDLER entity informs that a *stuff bit* was received, the CAN RX ignores this bit. So, it is not included in the stored message.

After receiving the Data Field, CAN RX informs to the CRC entity that the CRC sequence calculation of the received fields needs to be finished. At this moment, the CRC sequence reception is started.

When the CRC sequence is received, CAN RX compares it with the CRC sequence calculated. If CRC is correct, CAN RX acknowledges the message, sending a dominant bit.

2.3 STUFF HANDLER entity

This entity verifies if a stuff bit is required. If five consecutive equal bit levels are transmitted, STUFF HANDLER informs to CAN TX entity that a complementary bit needs to be sent. Moreover, when a stuff bit is received, STUFF HANDLER informs to CAN RX entity that this bit has to be ignored. This procedure is key to detect network failure.

2.4 CRC entity

This entity carries out the CRC evaluation of sent and received messages. The algorithm used in the calculation is in accordance with reference [2].

3. SIMULATION AND SHYNTHESIS

The described entities have been simulated and synthesized using the *ModelSim* (*Mentor®*) and *ISE* (*Xilinx®*) tools, respectively. Next, the VHDL CAN Module has been implemented using the *Xilinx® Spartan-II XC2S50* FPGA.

In this section, simulations of the CAN TX and CAN RX entities are discussed in detail. Next, the simulation of the completed interface, containing the CAN_TX, CAN_RX, STUFF_HANDLER and CRC entities is shown. After that, the transmission of a data frame using the VHDL CAN Module implemented on FPGA is observed. Finally, the synthesis results are analyzed.

The CAN TX waveforms are displayed in Figure 2a. After initialization, CAN TX waits until a message needs to be transmitted (START=1) and the bus is idle (RX_BUSY =0). At this moment, the Start of Frame bit is sent (TX_BIT = 0) and the TX_BUSY flag is activated. The Arbitration, Control and Data Fields are sent. After that, CAN TX requests the CRC Sequence (CRC_Request = 1). When the CRC Sequence is sent, CAN TX verifies if the message has been acknowledged.

Finally, the End of Frame Field is transmitted, TX_COMPLETED is activated and TX_BUSY is cleared.

The CAN RX waveforms are displayed in Figure 2b. After the reset cycle, CAN RX waits until a message starts to be received (RX_BIT = 0). At this moment, CAN RX starts storing the Message Identifier in MSG_ID. After that, the Control and Data Fields are received and the message content is stored in MSG.

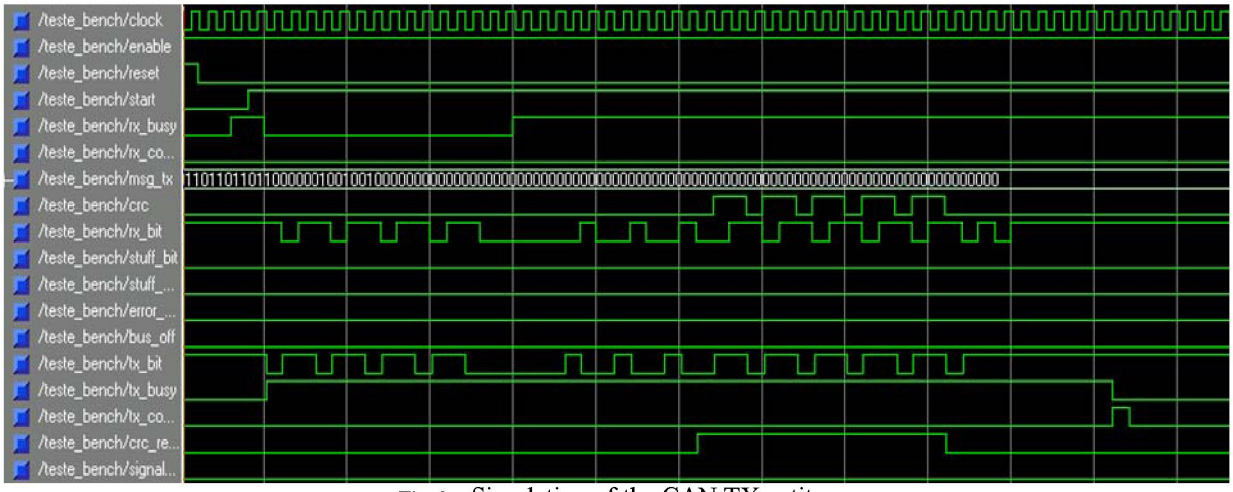

Fig. 2a. Simulation of the CAN TX entity.

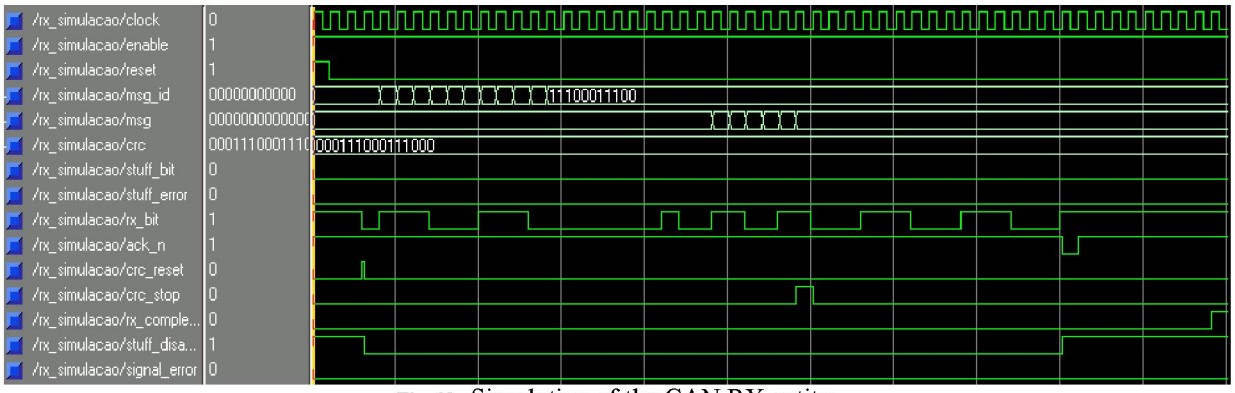

Fig. 2b. Simulation of the CAN RX entity.

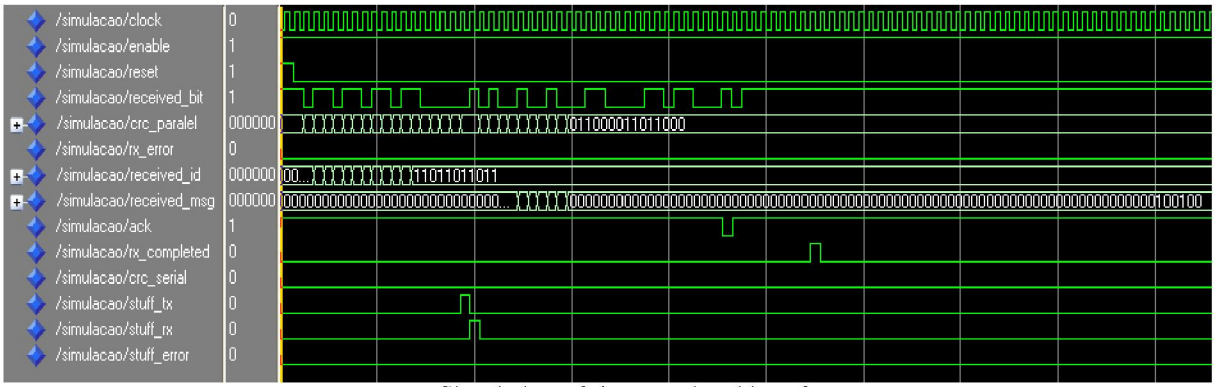

Fig. 2c. Simulation of the completed interface.

After the Data Field has been received, CRC_STOP is activated, informing to CRC entity that the CRC calculation needs to be finished. At this moment, the CRC sequence is received.

If the CRC Sequence is correctly received, CAN RX acknowledges the message, sending a dominant bit in ACK_N. Finally, the End of Frame is received and RX_COMPLETED is activated.

The simulation of the completed interface, containing the STUFF_HANDLER, CRC, CAN_TX and CAN_RX entities is shown in Figure 2c. As it can be observed, the STUFF_TX and STUFF_RX bits are generated by STUFF HANDLER entity after five dominant bits are transmitted and received. The CRC entity correctly performs the CRC sequence. Therefore, the message has been acknowledged and the transmission and reception ends.

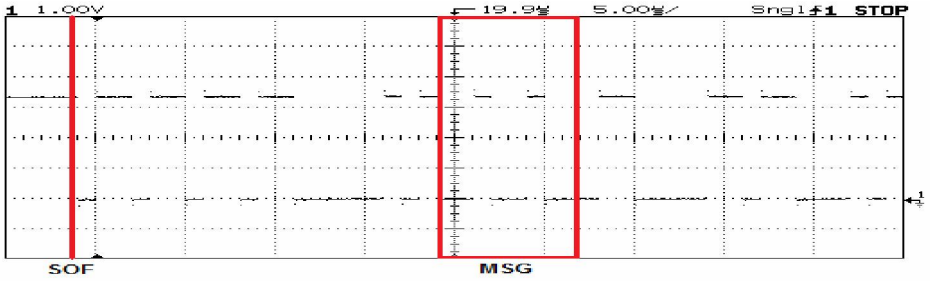

Fig. 3. Transmission of a data frame after the implementation on FPGA.

Table 1. Comparison between this paper CAN implementation and *HurriCANe*.

Logic utilizations	STUFF HANDLER		CRC		CAN TX		CAN RX	
	MARIA	HurriCANe	MARIA	HurriCANe	MARIA	HurriCANe	MARIA	HurriCANe
Slices	8	6	26	21	223	99	171	264
Slice Flip	7	6	31	31	154	113	153	258
LUTs	15	9	49	27	434	188	315	202
IOBs	8	6	23	23	117	148	105	148
GLCKs	1	1	1	1	1	1	1	1

The transmission of a data frame by this CAN module as implemented on FPGA has been verified with a digital oscilloscope. A sample screen image is shown in Figure 3, in this figure, the start of frame (SOF) and the received message (MSG) are marked. The data field is the byte 00100100.

In order to evaluate the allocation of resources in the developed design, a comparison between *HurriCANe* [3], a free VHDL CAN Controller Core, developed by ESA *(European Space Agency)*, and this implementation has been carried out. Although the *HurriCANe* Core used to be free, today, its code is not readily available anymore. For comparison purposes, the results in reference [4] are used.

In Table 1, a comparison between the described entities and the corresponding entities in *HurriCANe* is presented. The evaluation criterion used is the device logic utilization summary. The device logic utilization summary of *HurriCANe* can be found in reference [4].

As it can be observed in Table 1, the STUFF HANDLER entities uses approximately the same amount of resources as the corresponding entity in *HurriCANe*, although there is a small disadvantage with respect to the number of 4 inputs LUTs. The same occurs with the CRC entity. However, this CAN TX entity uses a significantly larger amount of resources than *HurriCANe* implementation. On the other hand, the CAN RX entity uses a lower number of resources in practically all logic utilizations, compared to *HurriCANe*.

4. CONCLUSIONS

This work presents the synthesis of a communication module for smart sensors which performs the transmission and reception functions, in accordance with the CAN protocol.

The described entities were simulated, synthesized and implemented on FPGA. It was observed that the designed entities functioned correctly. Further validation is being carried out.

A comparison with the freely available VHDL CAN Controller Core, *HurriCANe,* indicates the this receiver solution uses less FPGA resources. To improve the result of the transmitter, a RTL approach to get a more efficient area occupation and higher performance is under development. Next, this design will be integrated into a chip with *Mentor®* tools. As a further step, this module will be evaluated in a wireless environment.

5. REFERENCES

[1] *CiA - CAN in Automation*, www.can-cia.org.

[2] R. Bosch, "CAN Specification", *Robert Bosch GmbH,* Version 2.0, Parts A and B, Sept. 1991.

[3] L. Stagnaro, "HurriCANe – Free VHDL CAN Controller Core", *European Space Agency*, Mar. 2000.

[4] A. Amory and J. P. Júnior, "Sistema Integrado e Multiplataforma para controle remoto de residências", *Pontificia Universidade Católica do Rio Grande do Sul*, Dec. 2000.

[5] R. Stoneking, "A Simple CAN Node Using the MCP2510 and PIC12C67", *Microchip Technology Inc., Application Note*, 2002.

[6] S. Corrigan, "Introduction to Controller Area Network (CAN)", *Texas Instruments, Application Report*, 2002.