The TinyCAN: An Optimized CAN Controller IP for FPGA-Based Platforms

Fabiano C. Carvalho, Ingrid Jansch-Pôrto
and Edison P. Freitas
Computer Science Institute
Universidade Federal do Rio Grande do Sul
Avenida Bento Gonçalves 9500
CEP 90501-970, Porto Alegre - RS
{fccarvalho, ingrid, epfreitas}@inf.ufrgs.br

Carlos E. Pereira
Electrical Engineering Department
Universidade Federal do Rio Grande do Sul
Av. Osvaldo Aranha 103
CEP 90035-190, Porto Alegre - RS
cpereira@eletro.ufrgs.br

Abstract

This paper presents the TinyCAN, a resource constrained Controller Area Network controller implemented in VHDL language. The core description is fully synthesizable and fits in a FPGA-based platform occupying a maximum of 366 look-up tables. An enhanced error control strategy was elaborated with the aim of producing a more suitable behavior for safety-critical applications. For all that, compatibility with off-the-shelf components was preserved. Implementation details and comments on design decisions are given. The paper also presents some guidelines for the use of the proposed CAN controller IP for time-triggered architectures.

1. Introduction

The growing complexity of embedded systems has been forcing manufacturers to provide designers with complete integrated hardware platforms to accommodate all required functions. One of the main reasons for that is the highly heterogeneous nature of current and future distributed real-time embedded systems, embodying hardware and software components from various sources, including off-the-self and pre-designed legacy blocks. Its noteworthy that a new concept of embedded platforms is emerging. Nowadays, there are FPGA-based integrated solutions where programmable logic is included in addition to general purpose processors, allowing dedicated hardware to be synthesized according to the application needs. Besides reconfigurability features, low turnaround time of rapid prototyping using FPGA devices is an attractive alternative of system validation, specially when fast time-to-market is required. Equally important, FPGA technology is being largely used in final products when total demand is restricted to few units because of the high cost associated to ASIC fabrication.

Not only HDL developers but also the IP industry as a whole are aware of all those benefits. In the real-time embedded communication domain there are many available network controllers implemented in the form of synthesizable VHDL or Verilog modules. For instance, DECOMSYS[1] offers the so-called Automotive Prototyping Platform with customizable FPGA where a FlexRay controller can be synthesized. Another, BOSCH GmbH[2] offers a VHDL CAN controller IP[3] module for both Xilinx and Altera FPGA-based platforms. The obvious inconvenience is that in both cases HDL code and implementation details are proprietary so, they are of limited use for academic research.

Following these trends, this paper presents the TinyCAN, an open and fully synthesizable VHDL description of a CAN controller. To achieve a more reliable solution in terms of time composability, the TinyCAN core has slight differences with respect to the original protocol. An alternative error handling strategy was implemented in order to minimize the drawbacks of its utilization in safety-critical applications and to offer better support for future extensions. Besides that, compatibility with conventional off-the-shelf componentes was maintained.

This paper is organized as follows: section 2 briefly reviews main characteristics of the protocol and gives an overview on the error handling in CAN; section 3 describes the organization of the TinyCAN core and discusses some design decisions; section 4 gives some guidelines for development of a time-triggered extended platform; in section 5, synthesis results are presented and finally, in section 6, some concluding remarks are made.

2. Error handling in CAN

CAN is a serial bus communication system with broadcast and multi-access capabilities. It was originally developed for vehicle applications in the early 1980's by Bosch GmbH with the intent to provide a cost-effective solution for ECU (Electronic Control Unit) interconnection by solving the problem of excessive wiring. The specification of the protocol operation behavior and mandatory

[1] http://www.decomsys.com/
[2] http://www.can.bosch.com
[3] Intellectual Property

features is dictated by the ISO-11898 standard which covers the two lowest layers of the ISO OSI reference model. Besides of being widely used in automotive applications, the CAN protocol has also become largely used in factory automation.

Unfortunately, CAN is not adequate for safety-critical missions. The most relevant drawbacks have already been recognized by several authors such as [4] and [6]. Typically, schedulability analysis of data traffic on peak load scenarios, e.g., when the transmission of several messages is requested at the same time, is too optimistic since the occurrence of errors is not considered in most of the cases. In hazardous environments, where burst and transient network errors are more frequent, the response time for transmission of low priority messages could reach prohibitive upper limits due to successive error signalling and automatic retransmissions.

The CAN protocol includes several error detection mechanisms to provide best-effort reliable message delivery. Transmission errors are monitored by the bit error and ack error mechanisms while reception errors are detected either by stuff, CRC or form error mechanisms. In all cases, a specific bit sequence, which is called an error frame, is transmitted in the bus to inform all nodes that an isolated error has occurred in an active node. In sum, an error frame is basically a bit sequence that forces violation of stuff coding by transmitting more than five bits of the same polarity.

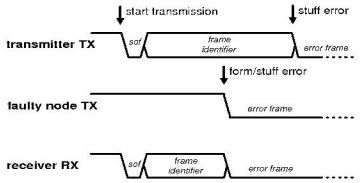

Figure 1. Faulty Node Disturbance

Nevertheless, a difficulty that comes out is that a node with a permanent failure can disrupt communication between non-faulty nodes. Consider the example shown in Figure 1. There, a transmitter node tries to send a data frame to a non-faulty node that needs this information. A third node which has, for instance, a failure on its incoming link, starts transmitting an error frame that overlaps the data frame at the moment the error is detected. It follows that retransmission attempts will take place as long as the failure persists or the internal controller's error counter of the faulty node does not reaches its upper limit forcing a transition to passive mode. In [4], the authors consider the impact of network transient and burst errors in CAN and conclude that, in safety-critical control applications, not to receive a message is better than to receive it late, otherwise it would prevent many other messages from arriving on time at destination. Strictly speaking, retransmissions must be disabled. The problem related to error frame generation becomes even more severe in time-triggered extensions to CAN based on a TDMA arbitration scheme, for instance [3] and [2]. An error frame followed by a not expected retransmission will certainly overlap more than one time slot, probably causing synchronization loss.

Our approach assumes that not only retransmissions but also error frame generation must be eliminated. In the TinyCAN core, all error detection mechanisms are present but there are no error counters neither error frame control logic. As a result, the core provides a more suitable error containment strategy. It adopts a fail-silent behavior on which errors are detected but not signalled.

3. The TinyCAN Core

The internal organization of the TinyCAN core is depicted in Figure 2. The **Clock Scaler** is a clock frequency divider from which the time quanta is derived. All significant events inside the controller are triggered by the time quanta **tq_ck** signal, including state transitions of the **Bit Time Logic** and **Bit Stream Processor** entities. The **Bit Time Logic** is the system interface with the physical media, it is responsible for the generation of the bit time which is constantly subject to resynchronization at recessive to dominant transitions during reception of a frame. Finally, the **Bit Stream Processor** in turn can be thought as the kernel of the controller, since it encapsulates basic functionalities, such as frames serialization and parallelization[4], error detection mechanisms and bitwise arbitration control.

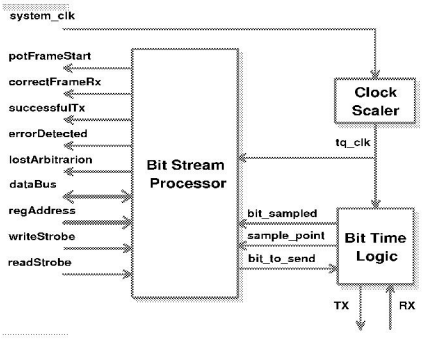

Figure 2. Simplified Block Diagram

A small four-register bank is used for network data flow exchange between the controller and the host CPU. Two registers of this bank are dedicated to load a single message for transmission while the others are used to store an incoming message. Whenever a low priority message is stuck in the transmit buffer and the application is waiting to send, it can be decided to overwrite the buffer's contents with the most urgent information at the moment.

In order to provide a better time composability characteristic throughout the whole system, an alternative error handling strategy was implemented. The TinyCAN has no control logic for error frame transmission (including overload frames), no error counters and no automatic retrans-

[4]The TinyCAN exchanges frames in the standard form only.

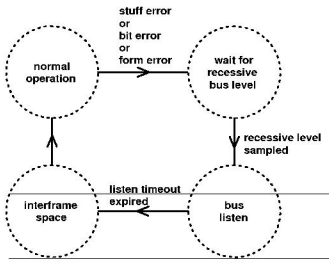

Figure 3. Error Recovery Procedure

missions. Once an error is internally detected, it adopts a fail-silent behavior and does not prevent other nodes from communicating by sending an error frame. Instead, the controller execute the recovery strategy shown in Figure 3. The **Bit Stream Processor** immediately makes a transition to a state where it waits until a recessive bus level is sampled. When this condition is met, it goes to a listen state, where there is no permission to transmit messages but a start-of-frame from another node is expected. The listen timeout is expressed in terms of bit times and is the maximum sequence of consecutive recessive bits allowed inside any CAN frame. Last, the original interframe waiting-state is reached and then normal operation is restored, either by sending a pending frame or going idle. Besides this subtle design decision, the TinyCAN is able to communicate with any other standard CAN controller. Since error and overload frames sent by an active node consists of a dominant bit sequence, they will be treated as a stuff error.

4. Time-Triggered Communication Support

Research on dependable distributed systems has recognized the time-triggered communication model as a adequate design paradigm for hard real-time applications. It is well-known that communication architectures derived from the time-triggered model have some desirable properties like low communication jitter, predictable transmission delays, time composability and efficient support for fault-tolerant functions.

4.1 Distributed Clock Synchronization

A time-triggered architecture depends upon the execution of a clock synchronization algorithm that periodically corrects the local time base in order to maintain it aligned/synchronized with the rest of the cluster within the limits of an attainable precision. Some kinds of distributed synchronization algorithms [1] lie on the dissemination of explicit clock values at the end of the resynchronization interval. Hereby, extra communication overhead is generated which restricts the available bandwidth for application data flow. Conversely, in TDMA based protocols [5], the values of remote clocks can be extracted from any frame arrival. Thanks to the static nature of the communication schedule it is possible to take the instant of a frame transmission start as a clock reading of its sender. As the clock readings are implicitly inferred, no additional information needs to be inserted in the frame. As a consequence, the total bandwidth is dedicated to application data flow.

The TinyCAN core is suited for the second class of synchronization techniques above. The **Bit Stream Processor** delivers the **potFrameStart** interrupt signal whenever a potential start-of-frame is detected on the bus. At this moment a timestamp from the local clock is taken as a possible representation of the sender's local clock. Then, if correct reception of that frame is confirmed by the controller, the clock read is validated and this value can be used by the clock synchronization algorithm to calculate a correction term to apply on its local clock.

4.2 Cluster Startup

Cluster startup is a fundamental low level service in a time-triggered communication scheme. It is basically the system's ability to create a global time base and to initiate the TDMA transmission schedule. When the cluster is energized for the first time there is no global time base so, collisions may occur during the initialization phase. The startup control service must be aware of that to accomplish its mission. For instance, a provably successful strategy is needed to avoid that collisions propagate indefinitely.

In a CAN-based time-triggered architecture, the solution is straightforward if the original non-destructive resolution mechanism is maintained. To explain how a TinyCAN cluster startup procedure would work, some useful concepts are introduced next. First, a **coldstart** is an attempt to initialize the global time and communication schedule. It is considered that not all nodes are authorized to perform a coldstart so, there are **coldstart nodes**, the ones that have permission to attempt to start the cluster, and **non-coldstart nodes**, the ones that have not.

Figure 4 shows a simplistic startup procedure of a cluster composed of 4 nodes. Nodes 1 and 4 are allowed to perform a coldstart by trying to send a **startup frame**, which contains additional schedule information. In this example, these two nodes are turned on at the same time hence, they listen to the bus and then decide that there is no ongoing communication activity. At this moment the CAN collision resolution mechanism comes into play. Since node 1 sends a higher priority message, it wins arbitration and becomes the **leading node**, e. g., the one that will create the global time base and initialize the scheduler. The other nodes (including the one that lost arbitration) will take a timestamp of the leading's transmission start instant and its schedule information to initialize their global parameters in turn. After successful transmission of the startup frame from the leading node, the clock synchronization algorithm starts execution and the TDMA arbitration scheme takes place.

The TinyCAN can be extended to a time-triggered communication platform with minimal effort to implement the startup control strategy. At a higher level, this procedure is simplified since collisions are resolved in the

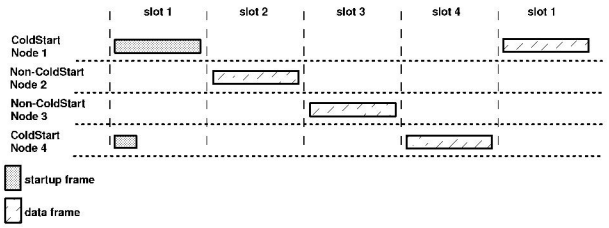

Figure 4. Startup Procedure

underlying data link layer that preserves the extremely efficient non-destructive resolution mechanism of the original CAN protocol. Another, this benefit may justify its utilization despite the maximum bitrate of 1Mbit/s, which is relatively slow.

5. Core Validation

The correctness of the TinyCAN core was verified by means of RTL simulation followed by a prototyping phase. Successful synthesis was achieved using a small amount of programmable resources, as indicated in Table 1. A total of 3 Xilinx Spartan2E FPGA target devices and a serial twisted cable were used to implement a network prototype. In all boards, the Tx and Rx signals from each FPGA device were mapped into I/O pins to interface with analog transceivers.

Table 1. FPGA Utilization (Spartan2E)

	Total Available	Used	Percentage
slices	2352	196	8%
slice Flip Flops	4704	95	2%
4 input LUTs	4704	366	7%

For the sake of validation only, the internal transmit flag was tied to TRUE in the VHDL code to force the core to believe that there is always a pending message to be sent. As a consequence, a collision always occurs. Figure 5 shows the scope images where the arbitration phase is clearly distinguished. The lower signal in both images is the Tx pin of the node that sends the higher priority message while the other is the Tx pin of a node that always looses arbitration. At the end of message transmission, the correct reception is confirmed by a dominant acknowledgment bit according to normal protocol operation.

6. Concluding Remarks

This paper presented the TinyCAN, which is a resource constrained CAN controller that implements a fail-silent behavior in case of transmission errors in order to overcome undesirable timing effects. Our main goal was first to make clear that it is worthwhile to provide conventional hardware components as synthesizable VHDL modules in view of the growing popularity of FPGA-based platforms. Second, its was described how the controller reacts when errors are detected, ensuring a better performance in terms

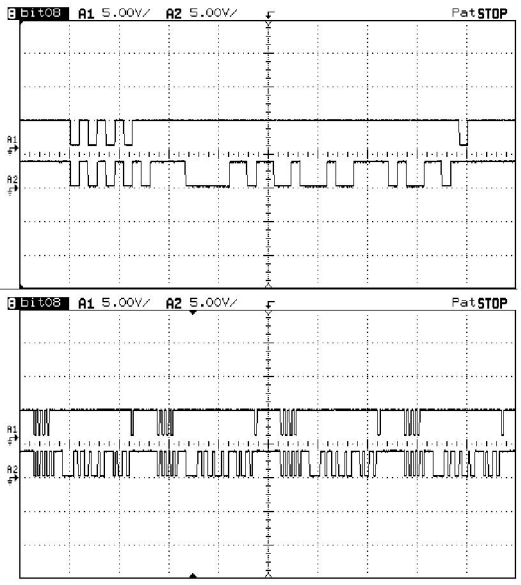

Figure 5. Collision Resolution

of reliability of the overall system as well as providing low level support for the implementation of a time-triggered extension.

References

[1] Rodrigues, L.; Guimaraes, M.; Rufino, J. *Fault-tolerant clock synchronization in CAN.* The 19th IEEE Real-Time Systems Symposium, 1998.

[2] Almeida, L.; Pedreiras, P.; Fonseca, J.A.G. *The FTT-CAN protocol: why and how.* IEEE Transactions on Industrial Electronics. Volume 49, Issue 6, Dec. 2002.

[3] Thomas Fürer; Bernd Müller; Werner Dieterle; Florian Hartwich; Robert Hugel; Michael Walther; Robert Bosch GmbH. *CAN Network with Time Triggered Communication.* Proceedings 7th International CAN Conference, 2000, Amsterdam.

[4] Broster, I.; Burns, A. *Timely use of the CAN protocol in critical hard real-time systems with faults.* 13th Euromicro Conference on Real-Time Systems, 2001.

[5] Kopetz, H.; Bauer, G. *The time-triggered architecture.* Proceedings of the IEEE Volume 91, Issue 1, Jan. 2003.

[6] John Rushby. *Bus Architectures for Safety-Critical Embedded Systems.* First Workshop on Embedded Software, 2001.