A Practical Implementation of the Fault-Tolerant Daisy-Chain Clock Synchronization Algorithm on CAN

Fabiano C. Carvalho
Instituto de Informática
Universidade Federal do Rio Grande do Sul
Centro de Excelência em Tecnologia
Eletrônica Avançada – CEITEC
Porto Alegre, Brazil
carvalho@ceitec.org.br

Carlos E. Pereira, Elias T. Silva Jr.,
Edison P. Freitas
Instituto de Informática
Universidade Federal do Rio Grande do Sul
Porto Alegre, Brazil
cpereira@ece.ufrgs.br
{etsilvajr, epfreitas}@inf.ufrgs.br

Abstract

Networked processing units are becoming widely used in the automotive embedded system domain aiming not only to reduce vehicle weight and cost but also to assist the driver to cope with critical situations. Because the fact that these embedded networked systems are strictly involved with human safety, there is a high demand on dependability requirements which can only be guaranteed if active redundancy is employed. Considering that the processing units are usually connected by a shared serial media, the underlying communication platform is the most important building block. It must provide low-level support for deterministic data transmission as well as a global time base to coordinate the actions of replicated units. Within this context, this paper presents the development of the fault-tolerant Daisy-Chain clock synchronization algorithm over the CAN protocol, resulting in an highly optimized communication architecture for safety-critical applications. Implementation issues and some obtained practical results are also discussed in the paper.

1 Introduction

As computers become increasingly more involved with healthy and safety of humans, dependability becomes a key factor when developing completely integrated solutions. It is noteworthy the growing interest in the automotive industry towards the replacement of mechanical artifacts by internetworked electronic units to perform dynamic control such as steering, braking, power-train and suspension control. Besides cost and weight diminution, the use of embedded computers inside vehicles should also assist the driver to cope with critical situations. This current trend has been forcing designers to cope with new requirements that are far beyond those of traditional distributed systems where there are no major risks in case of failure. Safety-critical systems must provide correct service delivery even in the presence of faults or any other unpredictable behavior since a single failure can lead to disastrous consequences, possibly resulting in loss of life or property.

Recently, it has been reported that failure rates less than 10^{-9} per hour are required in some particular cases [6]. However, that requirement can only be met if active redundance is employed combined with fault-tolerance mechanisms. Because replicated components have to make equivalent decisions at about the same time, redundancy coordination strongly depends on the correct operation of a clock synchronization algorithm to maintain a system-wide global time agreement. In view of that, clock synchrony must not be considered just an add-on feature of the system but an essential low level service of the underlying communication platform over which safety-critical functions are mapped.

In a dependable distributed system, the communication architecture must support straightforward integration of components and subsystems (including corresponding replica) without jeopardizing the temporal characteristics of each one individually. In other words, it must support time composability to ensure that one function does not adversely affect another under any possible operating conditions. From [6], [4] and [2] one can observe a common consensus indicating that this is only guaranteed if the channel bandwidth is fairly partitioned according to a time-division multiple access (TDMA) arbitration scheme, where each node transmits messages in dedicated time slots. By statically configuring all messages to be sent at pre-defined instants of a global time base, contentions for the bus will not occur so, even messages from low priority tasks will never be delayed.

3-9810801-0-6/DATE06 © 2006 EDAA

With all that in mind, we have implemented a highly optimized **C**ommunication **A**rchitecture for **S**afety-**C**ritical **A**pplications, or simply CASCA, that performs clock synchronization over the existing Controller Area Network protocol. The proposed CASCA architecture is composed of a logical and a physical block. The former one corresponds to a data link protocol plus extended time management functions while the second consists of a single (possibly replicated) serial bus together with COTS CAN transceivers. The logical block was entirely described in VHDL in order to take advantage of low turnaround time of rapid prototyping using FPGA devices, which becomes an interesting alternative of system design, specially when fast time-to-market is desirable.

This paper is organized as follows: section 2 discusses main characteristics of existing fault tolerant embedded protocols; section 3 briefly reviews the concepts of clock synchronization and existing algorithms, with special attention on the fault-tolerant Daisy-Chain approach; the proposed communication architecture is described in section 4; section 5 presents practical results and finally, in section 6, some concluding remarks are made.

2 Fault Tolerant Protocols

Embedded safety-critical applications consist mostly of control loops in which the execution of a control algorithm and data exchange from sensors to actuators take place at regular intervals within a known periodicity. In this domain, more important than throughput is the correctness of the results and the timeliness of data transmission services offered by the communication platform. In drive-by-wire applications, for example, the control frequencies have been estimated to be in the order of 100Hz [2]. If state messages are mainly of 32 bit length, then the total throughput for individual control loops is near 3.2kpbs which is relatively low compared to office local area network demands. Nevertheless, a primary requirement is that correct messages arrive on time when transmitted as they represent the system's view of the current state of the physical environment. Additionally, proper operation must be ensured even at the occurrence of faults since a late or corrupted state message can cause the system to compute wrong outputs and put human lives in danger.

Many years of experience and research has shown that the highest degrees of reliability in distributed systems are achieved if their entire design flow is guided by the concepts that surround the time-triggered paradigm, or simply TT. Roughly speaking, a TT distributed system is the one where actions are triggered as a global time base progresses. The matching list of trigger instants and corresponding actions is defined during the design phase and then inserted in a data structure inside each system network adapter.

The TT paradigm was first introduced by Hermann Kopetz who proposed the Time-Triggered Architecture (TTA) [4]. From the TTA three protocols derive: the TTP/A, TTP/B and TTP/C, which is the fault-tolerant version. The last letter of each protocol name indicates for which class of application it is intended for according to the Society of Automotive Engineers (SAE) classification of vehicle applications, covering from body electronics (classes A and B) to vehicle dynamic control (class C).

Bus arbitration in TT systems is basically performed by a time-division control logic which is tightly coupled with a low level clock synchronization algorithm. There is a sequence of TDMA slots where each node transmits one message thus forming a **round**. After finishing one TDMA round, the next one is started. The temporal access pattern of the new round is basically the same, but possibly different messages are sent. The number of different rounds determines the length of a **master cycle**. After a master cycle is finished, the transmission pattern starts over again. This cyclical operation behavior is maintained as long as all nodes keep synchronized one to another.

From the static communication behavior of TT systems several advantageous characteristics comes out. For instance, error detection is increasingly facilitated at both the sender and receiver sides due to the the fact that transmission and arrival times of each message are previously defined. Additionally, time composability is guaranteed as there are no bus contentions in a TDMA arbitration scheme. Finally, system verification and testing are both facilitated as a result of restricting the overall combinations of system states.

Actually, the time-triggered paradigm is being used as the primary design concept of the great majority of embedded protocols intended for safety-critical functions. The FlexRay protocol [1] from BMW and DaimlerChrysler is time-triggered because part of the communication bandwidth is dedicated to the transmission of pre-scheduled messages in a TDMA-based scheme. The CAN protocol, which has been largely used in the automotive industry for many years, is not suitable in terms of safety mainly due to the unpredictable variability of message transmission times. In the original CAN protocol, there is no global view of time. Any attempt to access the bus is triggered by asynchronous events from the host or from the environment in contrast to time-triggered architectures where all transmissions are scheduled at design phase. It follows that collisions eventually occur delaying the transmission of low priority messages. As a consequence, time composability can not be always guaranteed because individual functions may experience varying temporal behavior due to the shared use of communication media. In response to that, researchers from Bosch GmbH have created the TT-CAN (Time-Triggered CAN) [3] which is a time-triggered exten-

sion of the unchanged CAN protocol that is in the process of standardization by ISO as ISO 11898-4. Communication in the TT-CAN is organized in several TDMA rounds each one being triggered by a centralized master which broadcasts the reference message.

Both TTP/C and FlexRay operates over an uncentralized time base which is better in terms of reliability. Assuming that all nodes are fail-silent, any node that eventually fails will not prevent the others from communicating. On the other hand, the master-triggered approach adopted in TT-CAN requires that the master node is available all the time.

3 Clock Synchronization

In uniprocessor systems, synchronization mechanisms can derive directly from the system clock providing means to coordinate multiple execution flows and I/O accesses. Unfortunately, in a distributed processing environment the problem is indeed more complicated because physical imperfections of oscillator devices would create prohibitive divergences between execution speed of distributed tasks. The main reason is that crystal oscillators may generate an absolute error as large as 10^{-5} per second, which is equivalent to 0.86 seconds per day [7].

Informally, the objective of clock synchronization in distributed systems is to keep logical clocks of distributed processors approximately close to each other despite of that slight drifting apart of physical devices. Let i and j be two non-faulty nodes and $C_i(t)$ and $C_j(t)$ their corresponding logical times at any instant of perfect time t. It comes that if the following condition is satisfied:

$$|C_i(t) - C_j(t)| \leq \delta \tag{1}$$

for all i and $j \neq i$, then one can say that the system has a global view of time. The term δ is the precision of the global time base, that is, the maximum skew between non-faulty clocks.

Assuming that all logical clocks are initially within δ, clock synchronization is performed like follows.

1. each node broadcasts its logical clock to all other nodes
2. each node assembles the clock readings from other nodes and then calculates the correction term
3. at a specific point in time, all nodes apply the correction term at their logical clocks

Clock synchronization algorithms for TT systems differ mainly in steps 2 and 3. In FlexRay, the Fault-Tolerant Midpoint (FTM) algorithm is executed from the clock readings collected during the current and last TDMA rounds. At each execution of FTM, the k largest and k smallest values are discarded then the correction term is determined by the mean of the largest and the smallest of the remaining values [8]. The TTP/C protocol performs clock synchronization by applying the fault-tolerant average (FTA) algorithm on the most recent deviation values stored in a push-down stack of depth four. The correction term of FTA is the average of the two remaining values after discarding the smallest and the largest ones.

Another interesting solution is the Fault-Tolerant Daisy-Chain algorithm which was first proposed by Henrik Lönn [5] as an cost-efficient way for synchronizing clocks without appealing to a single, centralized clock source. Unlike FTA and FTM, the Daisy-Chain is simpler to implement since there is no need to temporarily store clock readings so, hardware complexity is reduced considerably. All nodes adjust their clocks according to the current transmitter's view of global time.

At any time, the maximum skew between non-faulty nodes provided by the Daisy-Chain algorithm is bounded by:

$$\delta_{max} = \epsilon + 2\rho R \tag{2}$$

where ρ is an upper bound of physical clock drift, ϵ is the reading error resulting from variable propagation delays and R is the synchronization interval, that is, the distance in time between two clock corrections. In practice, the actual runtime drift is unpredictable but one can make the assumption that it always lies within $\pm\rho_{max}$.

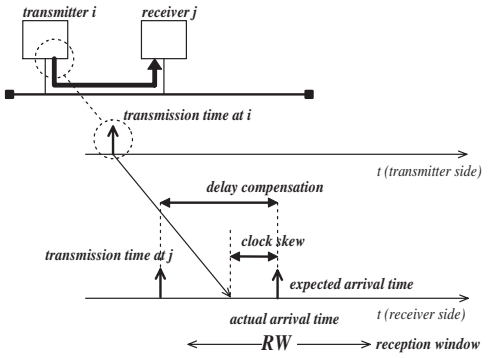

Figure 1. Clock reading in TDMA protocols.

The Daisy-Chain algorithm uses the principle of indirect clock reading, that is, the logical time of each transmitter is inferred at the receivers from the difference between the ***expected arrival time*** and the ***actual arrival time*** of the message, as shown in Figure 1. At the moment the transmission of a start bit from ***transmitter i*** is detected at ***receiver j***, that difference corresponds to the current ***clock skew*** between them. During normal operation, that skew is immediately used to adjust the local clock at ***j***. The definition of the expected arrival time must account for network propagation delays otherwise all receivers would have the impression

that they are ahead in time because the start bit arrives always later than expected. Rather than defining the expected arrival time at the exact ***transmission time at j***, it is repositioned at a later point by adding a ***delay compensation*** constant which is an accurate estimation of how long the signal takes to propagate from the source to destination.

For the Daisy-Chain algorithm to provide fault-tolerance against transient faults, clock readings whose absolute skew is too large are not accepted. The actual arrival time of any message must be approximately close to the expected time else its corresponding timestamp is discarded. Assuming that δ_{max} is the maximum skew between any two non-faulty clocks, the size of the ***reception window*** around the expected arrival time is defined as $2\delta_{max}$ in order to account for worst cases.

4 Practical Work

The logical block of CASCA that comprises the communication protocol with clock synchronization was entirely described in synthesizable VHDL code. The core of the system is a PIC processor that supervises all data flow between the components attached to an internal parallel bus. This strategy provides best error containment since a faulty component can not adversely affect another one without confusing the central core. Error detection mechanisms can be implemented in software to run at the PIC processor in order to fast isolate any component that eventually fails. If the processor fails itself, care must be taken to ensure that it assumes fail-silent behavior.

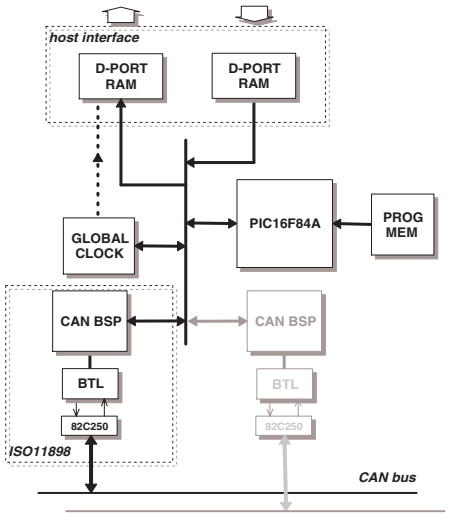

Figure 2. Internal organization of CASCA.

The underlying data link layer of CASCA is in accordance with the ISO-11898 standard that defines the CAN protocol, except that automatic retransmissions and error frame generation were not implemented in order to eliminate the possibility of slot overlapping in case of bus errors. The bit stream processor (BSP) encapsulates all logic for serialization and parallelization of CAN frames, checksum, stuff coding and bitwise arbitration control. The bit time logic (BTL) performs bit time synchronization throughout the reception and transmission of messages. It is responsible for maintaining the sample point at near 60% from the beginning of every bit transmitted. To employ bus redundancy, the entire data link layer can be replicated by attaching another instance of BSP and BTL to the internal bus, as suggested in Figure 2.

There are also two dual-port static RAM memories attached to the internal bus consisting of an interface with the host where state messages and control information are constantly updated. Despite the fact that the CASCA is essentially time-triggered, the host can also transmit asynchronous messages inside its dedicated slot whenever there is no critical data to be sent.

4.1 Clock Control

The Global Clock component encapsulates the control logic for communication time management. Prior to execution, the Global Clock is previously loaded with the static schedule information that is essential for protocol operation like slot size, number of slots per TDMA round and so on. The basic unit of time is called a *microtick*, extracted directly from the primary clock source, e.g. oscillator device. The length of a *microtick* does not need to be the same within all nodes since hardware platforms can have different oscillator device frequencies. Instead of that, at each node an integer number of *microticks* must be configured to generate the *macrotick* time unit. The length of the *macrotick* is a global parameter and it represents the minimum granularity of the synchronized time base. Its size must be an accurate estimation of δ_{max} for the system to have a consistent view of time.

As the *macrotick* counter progresses, interrupt signals from the Global Clock trigger significant actions to be executed by the PIC core such as the transmission and reception of messages. The *macrotick* counter is incremented up to the end of each TDMA round. When it reaches its upper limit, it restarts over again to execute a new TDMA round. In CASCA, clock correction is performed by the Daisy-Chain algorithm as described in section 3. The BSP component raises an interrupt signal whenever a start of frame bit is detected within the reception window. At this time, the PIC processor takes a snapshot of both *microtick* and *macrotick* counters and computes the difference from the expected arrival time. Once the message is correctly received, that difference is added to the *microtick* counter thus adjusting its local time base according to the sender's time.

The total propagation delay $\vec{P}_{ij}$ used to determine the expected arrival time of clock readings is defined as the sum of 3 components[1]:

$$\vec{P}_{ij} = \Delta_{Tx} + \Delta_{bus} + \Delta_{Rx} \quad (3)$$

where Δ_{Tx} is the output delay, Δ_{bus} is the bus delay proportional to the cable length and Δ_{Rx} corresponds to the input delay. Both Δ_{Tx} and Δ_{Rx} results from interrupt latencies and execution cycles so they can be exactly determined by simulation. On the other hand, Δ_{bus} is a non-uniform component because the cable length between each pair of nodes in a bus topology may vary. The best solution would be to statically define different compensation values which is exactly how a TTP/C system is configured. However, this would not only increase hardware complexity but also the amount of memory resources that would have to be allocated in the CASCA adapter. Instead of pursuing this path, it was decided to extract maximum and minimum bus delays and to use the average as a global parameter for Δ_{bus}, that is:

$$\Delta_{bus} = \frac{\Delta_{bus.max} + \Delta_{bus.min}}{2} \quad (4)$$

From this approximation the maximum error of a clock reading is bounded by:

$$\epsilon = \frac{\Delta_{bus.max} - \Delta_{bus.min}}{2} \quad (5)$$

The resulting value of ϵ must be used to compute the maximum drift apart of logical clocks according to equation 2.

4.2 System Initialization

Once the system is first energized, no assumption can be made about initial state of logical clocks so, a provably correct clock initialization procedure is needed at this moment to bring all clocks close to δ_{max}. The PIC processor is in charge of this task whose objective is either to force the node to adopt an existing communication schedule or to create a new global time base without the need to rely on a centralized time source.

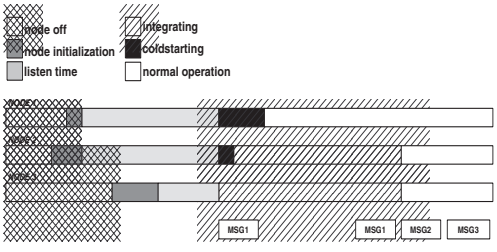

Figure 3. Clock initialization procedure.

[1]In star topologies the hub delay must be included.

Figure 3 shows how clock initialization works. When any node is powered on, it first listens to the bus in order to detect ongoing communication. If a message is received during this listen time then the node adjusts its counters according to the contents of the corresponding message identifier that carries the current slot and TDMA round. On the other hand, if there is no bus traffic then the node goes to a coldstart mode on each a new time base is created. The node that reaches its pre-allocated timeslot first will be the leading node by broadcasting a message with its own view of the time. By receiving the leader message all other nodes set their time variables according to its own schedule. After receiving two correct messages from the leader, all nodes start sending messages in their respective time slots. Since the clocks are not initially synchronized, collisions may occur if two or more nodes reach their slot time together. When that happens, the node with the highest priority node becomes the leader without perceiving that a collision has occurred, thanks to the original non-destructive bitwise arbitration of CAN.

5 Practical Results

A network prototype consisting of 3 nodes and a serial twisted cable was implemented using Spartan2E development boards. Table 1 shows the resource utilization synthesis report extracted from Project Navigator tool. Compared to the original CAN protocol, the extended time-triggered layer consisting of the Global Clock block and the PIC core represents an overhead of near 70% of programmable resources required to implement the CASCA logical block.

Table 1. Resource Utilization of the Xilinx Spartan2E Device

Block Name	slices	%	slice FFs	%	LUTs	%
CAN data link	315	27	239	33	556	26
Global Clock	353	29	140	19	637	30
PIC Core	527	44	346	48	948	44

In order to verify the correctness of the bit stream processor, the internal transmit flag was tied to TRUE in the VHDL code to force successive collisions. The bitwise arbitration can be clearly distinguished in Figure 4 showing a scope image of Tx signals at two nodes. Thereafter, each CASCA adapter was configured with the same TDMA schedule consisting of 5 TDMA slots and one single TDMA round pattern with total duration of 2ms. The *macrotick* length, that is, the precision of the global time was set to 1us considering oscillator drifts of 10^{-5} and maximum synchronization interval qual to the duration of a TDMA round.

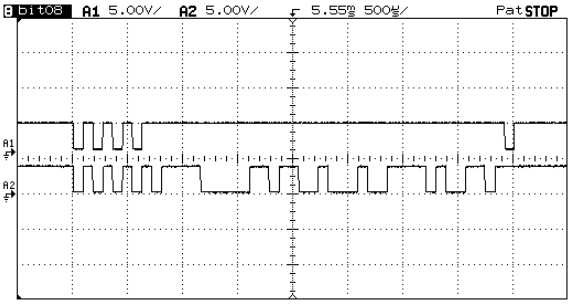

Figure 4. Non-destructive bitwise arbitration.

Figure 5 shows stable communication just after the startup phase in which two nodes have participated. Figures 6 and 7 shows messages from all nodes being sent after the third node has been successfully integrated to the current global schedule. For simplicity, all nodes send a CAN message of 16 bits at corresponding slot.

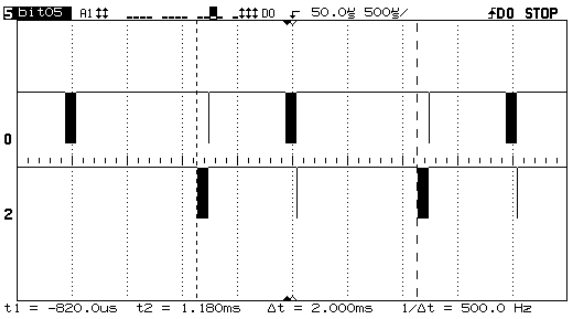

Figure 5. Two nodes online, one ready to integrate.

6 Final Remarks

The paper presented CASCA, an optimized, time-triggered communication architecture for safety-critical applications. Correct operation of protocol behavior and clock synchronization was experimentally validated by means of fast prototyping using FPGA devices. The CASCA logical block was entirely implemented in VHDL RTL code however, off-the-shelf CAN controllers can also be used if the following features are provided: 1) error frames and automatic retransmissions can be disabled; 2) the controller is able to raise an interrupt signal whenever a start of frame bit is detected and 3) input and output delays are deterministic. As future work, the authors are involved with the development of synthesis and modelling tools aiming to facilitate the mapping from system functional specification to

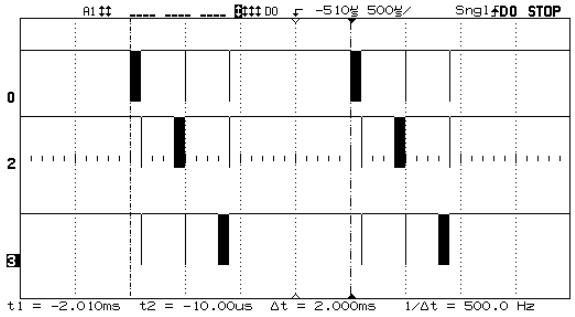

Figure 6. Three nodes online.

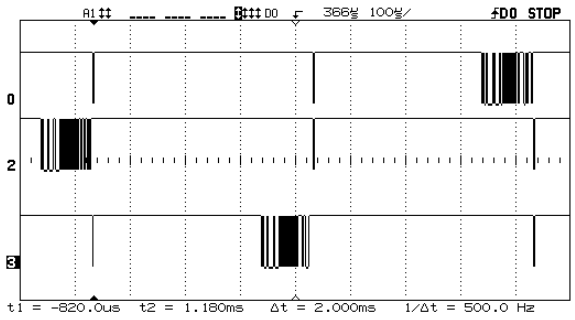

Figure 7. Three nodes online (zoom view).

hardware platform considering non-funcional requirements of fault-tolerance.

References

[1] FlexRay Protocol Specification. Technical report, FlexRay Consortium, 2004.

[2] O. Bridal, R. Snedsbøl, and L.-A. Johansson. On the Design of Communication Protocols for Safety-Critical Automotive Applications. *Proceedings of the IEEE 44th Vehicular Technology Conference*, 34:1098–1102, June 1994.

[3] T. Fuhrer, B. Müller, W. Dieterle, F. Hartwich, R. Hugel, M. Walther, and R. B. GmbH. Time Triggered Communication on CAN. *Proceedings of the 7th International CAN Conference*, 2000.

[4] H. Kopetz and G. Bauer. The Time-Triggered Architecture. In *IEEE*, volume 91, pages 112–126, January 2003.

[5] H. Lonn. The Fault Tolerant Daisy Chain Clock Synchronization Algorithm. Research report, Chalmers University of Technology, 1999.

[6] J. Rushby. Bus Architectures For Safety-Critical Embedded Systems. In *EMSOFT*, 2001.

[7] N. Suri, M. M. Hugue, and C. J. Walter. Synchronization Issues in Real-Time Systems. *Proceedings of the IEEE*, 82(1):41–54, January 1994.

[8] J. L. Welch and N. Lynch. A New Fault-Tolerant Algorithm For Clock Synchronization. *Inf. Comput.*, 77(1):1–36, 1988.