2021 17th IEEE International Conference on Factory Communication Systems (WFCS) | 978-1-6654-2478-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/WFCS46889.2021.9483588

# AUTOSAR-compliant Clock Synchronization over CAN using Software Timestamping

Florian Luckinger
*Elektrobit Austria*
Vienna, Austria
florian.luckinger@elektrobit.com

Thilo Sauter
*Institute of Computer Technology*, *TU Wien*, Vienna, Austria
*Dep. of Integrated Sensor Systems*, *Danube Univ. Krems*
Wiener Neustadt, Austria
sauter@tuwien.ac.at

***Abstract*—Despite its age and even though Ethernet is becoming popular, CAN is still widely used in distributed automotive systems. In such applications, a network-wide notion of time is often a prerequisite. However, precision clock synchronization over CAN is difficult to attain, and current approaches use dedicated hardware or proprietary software solutions. On the other hand, there is a standardized synchronization method which was defined by the AUTOSAR development alliance. This paper investigates the achievable precision using a purely software-based implementation of the AUTOSAR method using standard CAN controllers in a typical automotive real-time operating system. Preliminary results show that a precision of around 400 microseconds can be achieved with a fully AUTOSAR-compliant software implementation, and that there are options for further improvement.**

## I. Introduction and Background

Even though Ethernet is gaining importance in the automotive sector [1], CAN is still the predominant networking solution to connect electronic control units (ECUs), sensors, and actuators for classical applications with real-time requirements such as ABS, braking systems, or traction control. Like in all distributed systems, processes must be aligned properly in automotive applications, too, which requires the synchronization of the local clocks at the individual network nodes. In Ethernet-based systems, clock synchronization typically is done via the Precision Time Protocol (PTP) specified in IEEE 1588v2, which relies on round-trip delay measurements to achieve high accuracy. For CAN, the AUTOSAR (AUTomotive Open System ARchitecture) alliance defined a simpler approach for a standardized solution which uses a conventional reference broadcast scheme [2].

No matter which synchronization protocol is used, and independent of the underlying communication network it is well known that the accuracy of the synchronization is essentially determined by the accuracy of the timestamps. Hardware timestamps generally yield better performance than software timestamps [3], but require far higher implementation complexity. In particular, they demand specialized hardware components and cannot build on standard microcontrollers. Such approaches have been explored for CAN with varying complexity [4] [5] [6].

The performance of software solutions, on the other hand, depends strongly on where in the software stack timestamps are drawn and how tasks are organized. Non-preemptive tasks have a high impact on the timestamping behavior [7]. Implementing timestamping and thus also the reception of CAN messages via interrupts seems possible a way out, and there are several research works using this strategy [8] [9]. In practical automotive real-time systems, however, interrupts are often undesirable because they make timing predictions difficult. For this reason, polling-based IOs are used in many systems, which brings new challenges for timestamp generation.

The goal of this paper is to explore software-only precision clock synchronization over CAN following the AUTOSAR standard in a typical automotive real-time operating system using strictly timed round-robin task scheduling without interrupts. Sec. II introduces the basic AUTOSAR clock synchronization principle, and Sec. III presents the system model identifying the relevant delay mechanisms. Secs. IV and V describe the timestamping and the clock synchronization concepts, respectively, while Sec. VI presents the experimental setup and discusses preliminary results. Sec. VII gives an outlook on future work.

## II. AUTOSAR Clock Synchronization over CAN

In this paper, we consider only local synchronization, i.e., the internal synchronization of a group of nodes, irrespective of the global time (which could be given, e.g., by GPS). We consider a distributed system with $N$ participants, each with its own clock $C_i$. In the ideal synchronized case, all $N$ clocks show the same value at any time. In real systems, however, deviations occur. A measure for the synchronization quality of distributed systems is the precision $\Pi$. It indicates the maximum deviation between the clocks in the system and can be defined as

$$\Pi = \max_{i,j \in 1,2,\ldots,N} |C_i(t) - C_j(t)| \,. \tag{1}$$

The clocks used in this work are adder-based. A signal is generated by an oscillator with a fixed frequency. This signal is used as a basis for the local clock, and with each oscillator tick a preset value is added to the current clock value. By changing the clock increment, the speed of the clock can be adjusted. This allows for compensating *rate* differences between clocks, in addition to setting the clock *state* to a given value.

AUTOSAR clock synchronization over CAN [2] is used to synchronize different control units in a car. The definition of the standard is based on the AUTOSAR architecture for automotive embedded systems. It uses an unidirectional master-slave protocol with statically defined roles. Compared to PTP, it is a simple protocol that is tailored to the characteristics of the CAN bus. It supports both CAN and CAN FD, which limits the maximum size of the user data to 8 bytes per message. The standard defines timestamping of messages by sending and receiving acknowledgements, which are done only in software. The time is originally corrected purely by state correction.

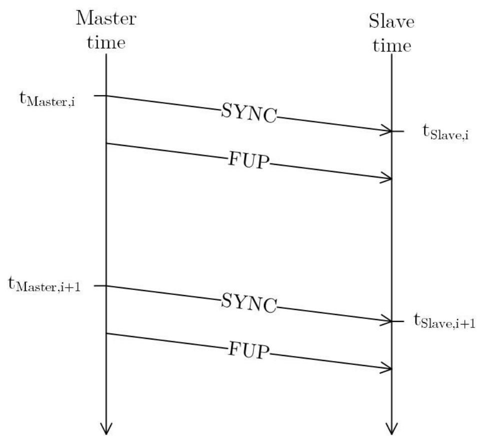


Fig. 1. Clock synchronization message exchange over CAN in AUTOSAR.

Fig. 1 shows the (simplified) message exchange. The master sends a SYNC message to the slave. This message is timestamped when sent by the master at $t_{\mathrm{Master,i}}$ and when received by the slave at $t_{\mathrm{Slave,i}}$. The master timestamp is subsequently transmitted using the FUP (follow-up) message. The two timestamps can now be used for synchronization.

## III. System Model

Fig. 2 shows a three-layer model of the system. The software layer comprises the processes that are executed on the CPU, which includes the application program for synchronization as well as the scheduler and the software drivers for reading out the CAN controller. The second layer includes the hardware components, such as the CAN controller itself. It provides buffers to the software layer for sending and receiving CAN frames. The third layer is the physical domain, comprising the transceiver and the bus line between the nodes. Following this model, the individual delays in message processing can be identified. The reference point $t$ marks the point at which a message is completely sent. The delay until the message is finally available in the application process is constituted by different parts of the system:

- $\Delta_{\mathrm{sched}}$ is the delay caused by the real-time scheduler. It describes the time span between receipt of the message in the message memory of the CAN controller and the callback in the software. Under ideal conditions, the maximum value of this delay is equal to the period of the receive task plus the software delays that occur when the callback is called. However, under real conditions this delay is larger, because the receive task can be delayed by other tasks.
- $\Delta_{\mathrm{read}}$ is the delay in reading the clock value. It is the time span between calling the read function and the actual reading of the timestamp. This value cannot be assumed constant because it depends on the execution time of the software.
- $\Delta_{\mathrm{ctr}}$, $\Delta_{\mathrm{trans\_rx}}$, and $\Delta_{\mathrm{trans\_tx}}$ are hardware-related delays associated with the sending and receiving of CAN frames until they are available in the message memory. They can be assumed as nearly constant because the CAN controller is completely implemented in hardware.
- $\Delta_{\mathrm{prop}}$ is the propagation delay of the signal over the CAN bus. It depends on the bus length and is constant for fixed network configurations.

These delay components also affect the software timestamping process. As will become clear in the next section, however, only the receive path is relevant for AUTOSAR-compliant implementation.

## IV. Software Timestamping Approach

The synchronization as well as the send/receive tasks run on the CPU and are invoked cyclically by the scheduler. When the send function is called, the data to be sent are placed in the send buffer and sent when the bus allows it. As soon as the message is in the send buffer, the send function returns. This time does not correspond to the time at which the message was actually sent, so it is not a suitable way to draw an accurate timestamp. When a message arrives over the bus, the CAN controller makes it available via the receive buffer if the processing was successful and no errors occurred during transmission. The software first notices a new message when the scheduler calls the CAN receive task. This task reads all messages from the receive buffer and notifies the applications.

The AUTOSAR clock synchronization over CAN standard requires that the timestamps are generated at the end of the frame. Frames that have been sent successfully appear the same for all bus nodes, including the sender. As soon as a sent message is received again, the sender knows that it was sent successfully. The send and receive timestamps can therefore be generated with the same mechanism. In both cases a message is received and as soon as the application is notified, the current clock value is read. The timestamps are given by

$$t_{\mathrm{Master,i}} = t + \Delta_{\mathrm{Master}} = t + \Delta_{\mathrm{sched}} + \Delta_{\mathrm{read}} + \Delta_{\mathrm{ctr}} + \Delta_{\mathrm{trans\_rx}} \quad (2)$$

for the master side and

$$t_{\mathrm{Slave,i}} = t + \Delta_{\mathrm{Slave}} = t + \Delta_{\mathrm{sched}} + \Delta_{\mathrm{read}} + \Delta_{\mathrm{ctr}} + \Delta_{\mathrm{trans\_rx}} + \Delta_{\mathrm{prop}} \quad (3)$$

for the slave. Apparently they differ only by the network propagation delay (which nevertheless cannot be determined in a reference broadcast scheme), however there is a significant influence by the stochastic components introduced by the software on both sides of the communication channel.

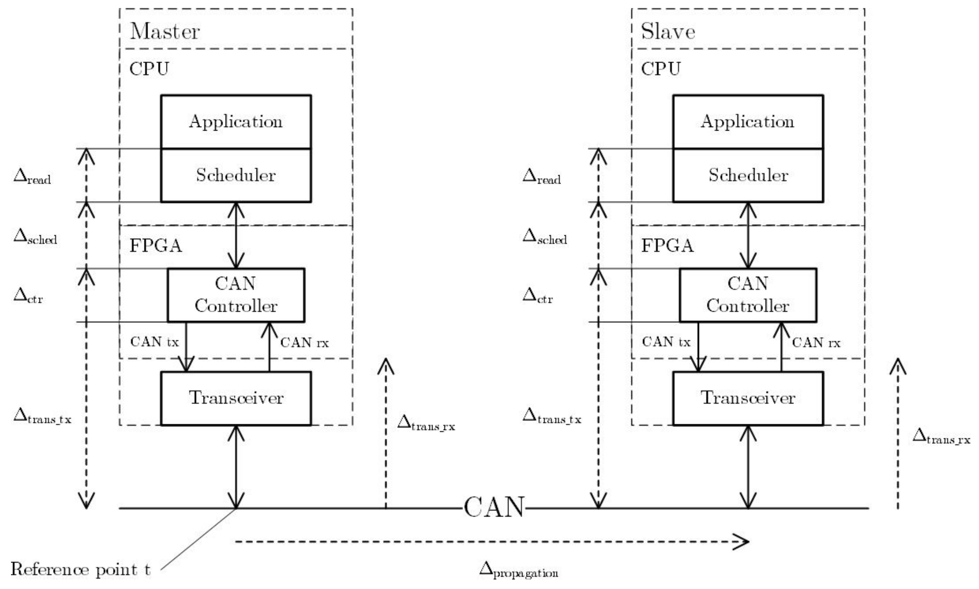


Fig. 2. System model indicating essential delay components. Only the receive path is relevant for clock synchronization.

## V. Clock Synchronization

With the timestamps the slave receives through the message exchange, the time can now be synchronized to that of the master. In the simplest case this happens by state correction, where the slave clock is set directly to the received value. This method has the disadvantage that time jumps can occur. Therefore in this work rate correction is used, where clock deviations are compensated by adjusting the rate of the clock. The mechanism employed here distinguishes between two rates. The *nominal rate* is syntonized to the rate of the master.

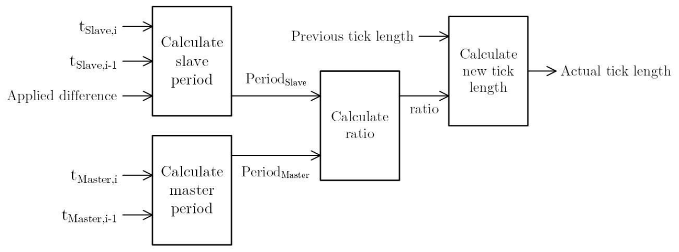


Fig. 3. Clock rate adjustment algorithm.

Fig.3 shows an overview of the basic model of the rate correction mechanism. The last synchronization interval is used to compute the momentary period for the slave, $P_{\text{Slave,i}} = t_{\text{Slave,i}} - t_{\text{Slave,i}-1} - c_{\text{offs}}$, and for the master, $P_{\text{Master,i}} = t_{\text{Master,i}} - t_{\text{Master,i}-1}$, where $c_{\text{offs}}$ accounts for a possible offset correction. The ratio $R_i = P_{\text{Master,i}}/P_{\text{Slave,i}}$ of the interval length at the master and slave is calculated and used to correct the current rate by adjusting the tick length, $TL_i = TL_{i-1} \times R_i$, i.e., the increment that is used for the added-based clock. For correcting the offset between master and slave a *temporary rate* is used. This is faster or slower than the nominal rate depending on the sign of the offset and is applied for a limited time. The duration is determined by the size of the offset.

## VI. Preliminary Experimental Results

The experimental setup shown in Fig. 4 uses the EBX200 hardware platform from Elektrobit consisting of a PowerPC processor and an Altera Cyclone V FPGA. The communication controller for the CAN bus is implemented in the FPGA and uses the Bosch MCAN IP core. Two such devices are synchronized with each other via CAN. One bit of each internal adder-based clock is routed to a general purpose I/O (GPIO) pin of the FPGA and connected to the oscilloscope. Therefore, the phase difference between the two signals is measured, which can be directly evaluated as offset of the two clocks. Apart from the synchronization messages exchanged every 3 s according to the AUTOSAR standard, not further bus load was considered, however the configuration of operating systems on the nodes was representative for real applications, and basic OS tasks like communication, memory management, or other system relevant functions were not optimized.

The evaluation is based on the assumption that ideally, the internal clocks of both devices should run synchronously and that the edges of the two signals occur at exactly at the same time. The maximum deviation of the edge timing can then be interpreted as the precision of the system. Fig.4 shows a screenshot of the measurements. The green line represents the clock signal of the master, the yellow line (this trigger signal) the one of the slave. Data were collected for about an hour. The results show a significant variability of the phase difference with a minimum of -360 µs and a maximum of 78 µs, with a mean value of -147 µs. This means that the slave clock lags behind the master as expected, and there remains an offset that cannot be detected by the AUTOSAR mechanism.

In the experiment, the precision $\Pi$ of the clock synchronization is 438 µs. This is better than other pure software-

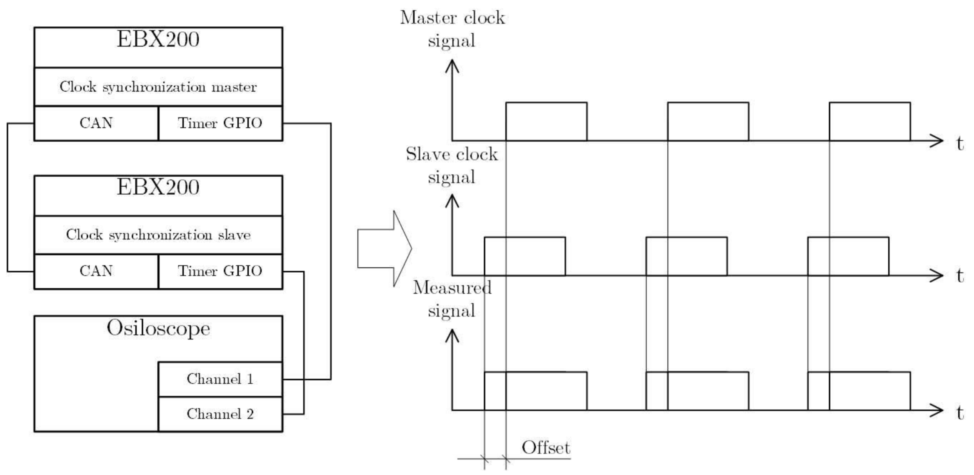
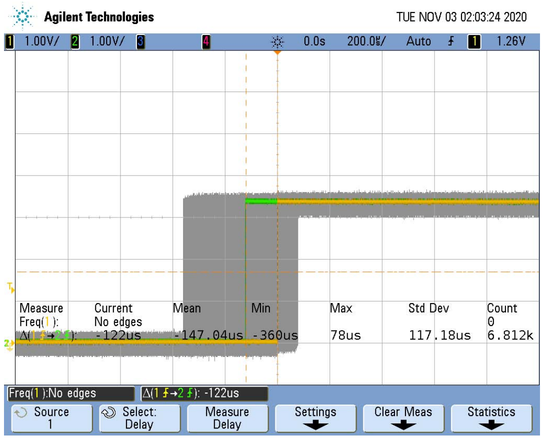


Fig. 4. Evaluation setup observing the local clocks at the master and slave node. The phase difference between the two signals is evaluated.

based approaches which reported precisions in the millisecond range [10] [8], but it obviously reflects the large uncertainty in the software timestamping which directly influences the clock rate adjustment. Notably, the precision is in the order of the scheduler period of 500 µs, which seems logical as the schedulers on both sides are not synchronized and message transfer is thus uncorrelated.



```
Measure      Current    Mean        Min      Max    Std Dev    Count
Freq(1 ):    No edges                                           0
Δ(1 ↑→2 ↑):  -122us     -147.04us   -360us   78us   117.18us   6.812k
```

```
Freq(1 ):No edges    Δ(1 ↑→2 ↑): -122us
```

Fig. 5. Precision of pure software-based synchronization.

## VII. Conclusions

Clock synchronization using only software timestamping has the benefit of not requiring special hardware, but is necessarily affected by scheduling uncertainties in the execution of relevant tasks as well as by the granularity of the scheduling period. The preliminary results reported in this paper clearly evidenced these limitations. However, there is room for improvement. One possibility is to reduce the scheduler period in order to reduce the quantization error that is expected to have a linear influence on the observed precision.

The second, probably more promising option is to use filtering to level out the strong delay variations involved in the timestamping process. In the current implementation, the clock rate adjustment is calculated only from the previous synchronization period. The mechanism in Fig. 3 could be amended by a filter applied to the calculated clock tick length. This would reduce the dynamics of the rate correction, which however would not be a disadvantage because the system setup is static. Such rate filtering might improve the synchronization precision by an order of magnitude. Future work will investigate these possibilities.

## References

[1] L. L. Bello, "Novel trends in automotive networks: A perspective on ethernet and the ieee audio video bridging," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, 2014, pp. 1–8.

[2] AUTOSAR, "Specification of time synchronization over can," Tech. Rep. 674 CP Release 4.3.1, 12 2017.

[3] A. Mahmood, R. Exel, and T. Sauter, "Impact of hard-and software timestamping on clock synchronization performance over ieee 802.11," in *2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014)*, 2014, pp. 1–8.

[4] G. Rodriguez-Navas, S. Roca, and J. Proenza, "Orthogonal, fault-tolerant, and high-precision clock synchronization for the controller area network," *Industrial Informatics, IEEE Transactions on*, vol. 4, pp. 92 – 101, 06 2008.

[5] G. Breaban, S. Stuijk, and K. Goossens, "Time synchronization for an asynchronous embedded can network on a multi-processor system on chip," in *2017 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, 2017, pp. 1–6.

[6] F. Hartwich, "Can frame time-stamping - supporting autosar time base synchronization," in *16th international CAN Conference (iCC 2017)*, 2017.

[7] J. Mitaroff-Szécsényi, P. Priller, and T. Sauter, "Compensating software timestamping interference from periodic non-interruptable tasks," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2017, pp. 1–4.

[8] P. Marti, M. Velasco, C. Lozoya, and J. Fuertes, "Clock synchronization for networked control systems using low-cost microcontrollers," Automatic Control Department, Technical University of Catalonia, Tech. Rep. ESAII-RR-08-02, 08 2008.

[9] J. Allan and D. Lee, "Fault-tolerant clock synchronization with microsecond-precision for can networked systems," in *9th international CAN Conference (iCC 2003)*, 03 2003, pp. 07–1 – 07–9.

[10] H. Daigmorte, M. Boyer, and J. Migge, "Reducing can latencies by use of weak synchronization between stations," in *16th international CAN Conference (iCC 2017)*, 2017.