DIPLOMA THESIS

AUTOSAR-compliant precision clock synchronization over CAN

Submitted at the Faculty of Electrical Engineering and Information Technology, Vienna University of Technology
in partial fulfillment of the requirements for the degree of Diplom-Ingenieur (equals Master of Sciences)

under supervision of

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Sauter Thilo

by

Florian Luckinger
Matr.Nr. 01426232

March 4, 2021
Kurzfassung


Abstract

The automotive sector is subject to constant change. Meanwhile, a single vehicle is a highly distributed system. For such systems, a common concept of time is particularly important for proper functioning.

For Ethernet based communication there is PTP for such tasks. This is the quasi-standard for high precision synchronization of clocks. In the automotive sector, however, the CAN bus plays a decisive role. However, there is still no standardized approach for clock synchronization. Existing solutions such as TTCAN require special hardware and change the communication scheme on the bus. The biggest problem with known solutions, however, is the precision achieved. While PTP can synchronize in the subnanosecond range, the best current solution for the CAN bus reaches 2 microseconds. This problem is addressed here.

The focus of this work is to develop a strategy for clock synchronization over the CAN bus that does not limit the choice of controller or communication scheme. The implementation is based on the AUTOSAR standard for clock synchronization over CAN. Only the message format is adopted, which means that the findings of this work can also be applied to other synchronization protocols. It has been shown that the way the timestamp is generated has the greatest influence on precision. However, even software-based timestamps could be improved by understanding the sources of error. The evaluation of the at the end of the work has shown that a precision in the two-digit nanosecond range is possible with the CAN bus.
Acknowledgements

This work was done in cooperation with the company Elektrobit Austria GmbH.
# Table of Contents

1 Introduction
   1.1 Goals of this thesis ........................................... 2
   1.2 Structure of the thesis .......................................... 2

2 State of the art
   2.1 Time representations ........................................... 3
   2.2 Clocks in computer systems .................................... 4
   2.3 Clock synchronization ........................................... 6
      2.3.1 Timestamping ............................................. 9
      2.3.2 Clock adjustment techniques ............................. 9
      2.3.3 Delay measurement .................................. 10
   2.4 Controller Area Network ..................................... 11
      2.4.1 Basic concepts .......................................... 11
      2.4.2 Messages ............................................... 12
      2.4.3 Bit timing ............................................. 13
      2.4.4 CAN FD .............................................. 14
      2.4.5 CAN XL .............................................. 14
   2.5 Precision time protocol ....................................... 15
      2.5.1 Basic concepts .......................................... 15
      2.5.2 Time distribution mechanism ............................ 15
      2.5.3 Best master clock algorithm ............................. 17
      2.5.4 Specifics ............................................. 17
   2.6 Autosar time synchronization over CAN ..................... 17
   2.7 Related work .................................................. 18
      2.7.1 Approaches with time triggered communication ......... 19
      2.7.2 Approaches with event based communication .......... 20
   2.8 Conclusion .................................................... 23

3 System overview
   3.1 Device structure ............................................. 25
   3.2 Hardware part ................................................ 25
      3.2.1 Bosch MCAN .............................................. 26
      3.2.2 Frame detection unit .................................... 27
      3.2.3 EB clock synchronization ip core ....................... 28
   3.3 Software part ................................................. 30

IV
3.3.1 Realtime core ................................................. 30
3.4 Elektrobit highspeed capture and replay protocol .................. 31
3.4.1 CAN frame ...................................................... 32
3.5 Relevant features for evaluation .................................. 32
3.5.1 Digital Event Triggered Input ................................. 33
3.5.2 Timer GPIO ....................................................... 33
3.6 Delay analysis ..................................................... 33
3.6.1 Software timestamping ....................................... 35
3.6.2 Hardware timestamping ...................................... 41

4 Approach .......................................................... 45
4.1 General model ....................................................... 45
4.1.1 Offset correction ................................................ 46
4.1.2 Rate correction ................................................ 50
4.2 Software solution .................................................. 52
4.3 Software solution with filtering .................................. 53
4.3.1 Averaging filters ................................................. 54
4.3.2 Exponentially weighted average ............................... 54
4.3.3 Moving median ................................................... 55
4.3.4 Kalman filter ...................................................... 56
4.3.5 Comparison ....................................................... 57
4.4 Improving software timestamping .................................. 58
4.5 Hardware solution .................................................. 60
4.6 Hardware solution with filtering .................................. 61
4.7 Delay measurement ................................................ 62

5 Evaluation .......................................................... 64
5.1 Measurement methods ................................................ 64
5.1.1 Measurement via timer GPIO .................................. 64
5.1.2 Measurement via DETI ......................................... 65
5.2 Measurement results ................................................ 67
5.2.1 Software solution ................................................. 67
5.2.2 Software solution with filtering ................................ 69
5.2.3 Improved software timestamping ............................... 70
5.2.4 Hardware solution ................................................. 72
5.2.5 Hardware solution with filtering ................................ 73
5.2.6 Hardware solution with filtering and delay compensation ....... 74

6 Conclusion and outlook ............................................... 80
6.1 Summary ............................................................ 80
6.2 Outlook .............................................................. 81

References ............................................................ 84
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOSAR</td>
<td>AUTomotive Open System ARchitecture</td>
</tr>
<tr>
<td>BMCA</td>
<td>Best Master Clock Algorithm</td>
</tr>
<tr>
<td>CAN</td>
<td>Controller Area Network</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
<tr>
<td>CSMA/CR</td>
<td>Carrier Sense Multiple Access/Collision Resolution</td>
</tr>
<tr>
<td>DETI</td>
<td>Digital Event Triggered input</td>
</tr>
<tr>
<td>EB</td>
<td>Elektrobit</td>
</tr>
<tr>
<td>EBHSCR</td>
<td>Elektrobit Highspeed Capture and Replay Protocol</td>
</tr>
<tr>
<td>FDU</td>
<td>Frame Detection Unit</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>GPIO</td>
<td>General Purpose Input Output</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>IP Core</td>
<td>Intellectual Property Core</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Systems Interconnection</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PCIe</td>
<td>Peripheral Component Interconnect Express</td>
</tr>
<tr>
<td>PPS</td>
<td>Pulse Per Second</td>
</tr>
<tr>
<td>PTP</td>
<td>Precision Time Protocol</td>
</tr>
<tr>
<td>TAI</td>
<td>Temps Atomique International</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TDMA</td>
<td>Time Division Multiple Access</td>
</tr>
<tr>
<td>TTCAN</td>
<td>Time Triggered Controller Area Network</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>UTC</td>
<td>Coordinated Universal Time</td>
</tr>
<tr>
<td>XMC</td>
<td>Switched Mezzanine Card</td>
</tr>
</tbody>
</table>
1 Introduction

The automotive sector is subject to constant change due to rapid technological progress, especially developments in electronics. In the beginning, cars were controlled by a central control unit. But due to growing demands, the development has moved away from this architecture. Today, it is common for a car to consist of a number of control units, which are typically divided into domains and connected. But due to growing requirements, especially in the area of automated driving, the architecture is evolving. The current trend is towards a zone architecture [EB221].

A car can therefore be seen today as a highly distributed system. And just as the architectures have changed over time, the same is happening for the communication interfaces that connect the individual subsystems. Whereas in the past the CAN bus was used exclusively, Ethernet is currently finding its way into the automotive industry. High-performance applications in particular are benefitting from this. But in the area with low performance requirements, CAN has always been a widely used system and there is no reason to expect this to change so quickly. On the contrary, CAN is still being further developed to meet current requirements. The next development stage is CAN XL and promises data rates of up to 10 Mbits [Aut20].

Another central point for distributed systems, besides the communication interfaces, is a common view of time. To ensure this, clock synchronization between the individual systems is necessary. For Ethernet-based devices, this is done via PTP [IEE20]. This standard provides a standard for the high precision distribution of a common time concept. The high precision is made possible by specialized hardware that can generate highly accurate timestamps independently of the software. However, such a standard procedure is missing for the CAN bus. There are some approaches like TTCAN, but these intervene deeply in the hardware and set a time-controlled communication scheme [FMD+01]. So special hardware is needed and the possibility of an event based communication is taken away. One possibility now would be to synchronize CAN nodes via PTP with a standard CAN controller. However, PTP is designed for Ethernet and therefore relies on very large messages. The standard CAN bus cannot meet this requirement because its user data is limited to 8 bytes. In addition, one of the biggest advantages of PTP is that many standard Ethernet controllers already offer hardware solutions for timestamping. This requirement is also not met by standard CAN controllers.

An approach to implement a PTP-like clock synchronization with CAN is provided by the development partnership Autosar [ASC17]. It defines a message format that is similar to PTP and does not require any special hardware or communication scheme. The method assumes that received CAN messages are immediately available for the software by interrupts, so that a software timestamp can be generated. However, this is not always the case for real-time systems,
since software execution times are often strictly controlled by a scheduler. This means that no high-precision clock synchronization as offered by PTP is possible, since no precise timestamps can be generated by the software. Thus, precision in the two-digit nanosecond range as offered by PTP is not possible here. A precision in this range can currently not be offered by any solution with the CAN bus.

1.1 Goals of this thesis

This thesis investigates strategies for clock synchronization over the CAN bus. The main goal is to achieve a precision of less than 100 nanoseconds. However, the solution shall not limit the choice of the communication scheme or the CAN controller, so that the solution can be used as independent as possible from the used system. Furthermore, the solution must not have any problems with common paradigms of real-time systems, such as strict task schedulers. To enable compatibility with other systems, the message format of the AUTOSAR solution should be adopted.

From these realizations the following problem definition results:

How can clocks be synchronized over the CAN bus with a precision of less than 100 nanoseconds without limiting the choice of the CAN controller or the type of communication?

1.2 Structure of the thesis

Chapter 2 deals with the basics of clock synchronization and current work on it. Basic terms are described and basic mechanisms are explained. Furthermore, the CAN bus and protocols used for clock synchronization are discussed. At the end of the chapter, current research work on the topic of clock synchronization via CAN is analyzed and tested for their applicability to this problem.

Chapter 3 is dedicated to the system on which the problem is implemented and evaluated. The components relevant for clock synchronization are examined and analyzed. Special focus is put on the type of timestamping and the related problems like delays. In addition, relevant components needed for the evaluation are described.

Chapter 4 deals with the development of strategies for high precision clock synchronization. A model based on the used system is created and refined step by step. Different filters are analyzed for their usability. Furthermore, the used standard is extended by a delay measurement.

In chapter 5 the used measurement methods are described and the measured results are discussed. Assumptions made in the chapter 4 are verified.

Chapter 6 summarizes the key findings of the thesis and gives an outlook on future extensions of this thesis.
2 State of the art

This chapter covers the basic standards and terms used in this work. It also analyzes the current state of the art in terms of current research.

2.1 Time representations

There are several global time standards that can be used as a basis for a system. This chapter is intended to give an overview of the most important representatives[MS18]. It also explains how they come about and how they relate to each other.

International atomic time

International atomic time is the primary physical time standard based on the SI-second. It is the basis for several other standards like UTC. It is usually abbreviated TAI, which comes from the French ”Temps Atomique International”. The time scale is achieved by averaging more than 600 atomic clocks from all over the world.

Coordinated Universal Time

Coordinated universal time, abbreviated UTC, is the primary astronomical time standard for clock regulation. The standard is based on the rotation of the earth and takes TAI as a basis for the duration of one second. Due to the slowing down of the Earth’s rotation, TAI and UTC are not synchronous. To compensate for this, leap seconds are introduced. Currently there are 37 leap seconds. The current relationship between UTC and TAI is therefore

\[ t_{UTC} = t_{TAI} - \text{actual leap seconds}. \] (2.1)

GPS time

GPS uses its own representation of time, called GPS time. This can be used for highly accurate global clock synchronization. It is based on TAI and was launched on 6 January 1980. At that time it was synchronous to UTC. At this time 19 leap seconds were already introduced. Therefore the relation is defined as

\[ t_{GPS} = t_{TAI} - 19s = t_{UTC} + \text{actual leap seconds} - 19s. \] (2.2)
2.2 Clocks in computer systems

In computer systems time is generated from oscillators. This chapter [Kop11] is intended to give an overview of how this is done, what its characteristics are and where possible sources of error lie.

Figure 2.1: Clock generation in computer systems

Figure 2.1 shows an overview of the generation of time in a computer system. This is typically divided into two domains. The physical domain, which consists of discrete circuits or dedicated devices, and the digital domain, which typically consists of microprocessors or FPGAs. In the physical domain, an analog clock signal is generated by an oscillator. This clock signal consists of periodic pulses with constant frequency. In the digital domain, the pulses are counted and thus converted into digital values. The resulting clock ticks are multiplied by a factor to obtain a representation of the real time. The resulting rate of change should be as close as possible to a reference time. As reference time for example UTC, TAI or GPS time can be selected. Thus it must correspond to

\[
\frac{dC}{dt} = 1, \tag{2.3}
\]

where the function \( C(t) \) corresponds to the physical clock. This function always outputs the counter reading at time \( t \).

The relationship 2.3 assumes a perfect oscillator with a stable rate. However, this is not the case with real oscillators. These change their rate depending on various factors such as temperature and aging. The resulting error is called clock drift and is defined as

\[
drift = \frac{C(t_{i+1}) - C(t_i)}{t_{i+1} - t_i}, \tag{2.4}
\]

the drift rate \( \rho \) is given as

\[
\rho = \left| \frac{C(t_{i+1}) - C(t_i)}{t_{i+1} - t_i} - 1 \right|. \tag{2.5}
\]

The figure 2.2 illustrates this relation. Perfect clocks have a drift rate of 0, therefore the relation 2.3 follows, for real watches the relation is calculated as
The time generated in computer systems can only take on discrete values. The smallest time unit that can be measured is called granularity $g$ and is defined as

$$g = C(t_{\text{clock tick},i+1}) - C(t_{\text{clock tick},i}),$$

where $t_{\text{clock tick},i}$ corresponds to the time of a clock tick.

Timestamps can be derived from a clock $C$, these describe the state of the clock at a time $t$. Due to the granularity, there is always a quantization error when determining timestamps. This error is illustrated in Figure 2.3.

This error can be as large as the granularity. Because of this connection two points in time can only be clearly distinguished from each other if they are separated by the amount of granularity.

The previous findings are based on the assumption that there is a clock with a reference time in the overall system. In distributed systems, however, there is typically more than one clock. Due to this fact, more parameters have to be considered.

Let’s assume we have two clocks $C_A$ and $C_B$. The absolute difference between the two clocks at time $t$, called offset, is calculated according to

$$\text{offset}_{C_A, C_B}(t) = |C_A(t) - C_B(t)|.$$
The maximum offset of all clocks in the system is called precision $\Pi$. This parameter is needed to determine the performance of internal clock synchronization.

Another important parameter for the performance of clock synchronization is the accuracy $A$, which is defined as

$$A = \max(|C(t) - t|).$$  \hfill (2.9)$$

It is the maximum difference of a Clock $C$ and the reference time in a given interval.

2.3 Clock synchronization

Clock synchronization [TVS13] is used to bring different clocks in a system to a uniform status. This state may depend on external sources, in which case it is called external synchronization. Such sources can be GPS or other high precision clocks. If all clocks in a system are synchronized to an external source, the system has the global time base of the external source. Since there are no perfect clocks in the local system and the rate is drifting, regular re-synchronization is necessary. If different clocks in a system are synchronized with each other without an external time source, this is called internal synchronization. Systems do not necessarily have to be synchronized internally or externally, because in practice mixed forms often occur. An example is a network of systems, one of which is equipped with a high-precision clock. This participant then synchronizes externally to this high-precision clock and distributes the time in the network by internal synchronization.

To achieve a certain precision it is necessary to synchronize regularly. The time in between is called resynchronization interval $R$. As soon as they have been synchronized, the clocks are in a defined state. The convergence interval $\Phi$ describes this additional time as remaining offset. After that the clocks drift apart again, the maximum offset that is created is called drift offset $\Gamma$. 

Figure 2.3: Timestamping error
This can be expressed by the drift rate of the local clock and by the resynchronization interval and is defined as

$$\Gamma = 2\rho R.$$  \hspace{1cm} (2.10)

This can be used to formulate the synchronization condition

$$\Phi + \Gamma \leq \Pi.$$ \hspace{1cm} (2.11)

This essentially states that the drift offset and the convergence interval represent the lower limit for the calculable precision. In figure 2.4 this relationship is shown graphically. The grey lines represent the maximum drift rate. Clocks with the precision $\Pi$ must remain within these lines.

![Figure 2.4: Synchronization condition](image)

This condition can easily be applied to simple synchronization algorithms. Most synchronization mechanisms are based on the fact that a master with a highly accurate clock periodically sends packets with its current time to participants. The participants take a timestamp when receiving the packet and compare it with the timestamp of the master. This results in a delay, which can be divided into different parts. To do this, a generic communication model is used, figured in 2.5. This model consists of two participants, a transmitter with a high-precision clock and a receiver. The transmitter periodically sends its local time to the receiver. The two participants each communicate via an unspecified communication medium with unknown delay.

For clock synchronization, the delay that occurs when a time is sent from one node to another is now relevant. The time-critical part of the process runs as follows:
1. A message is sent and a send timestamp is taken.
2. The message is transmitted.
3. The message is received and a reception timestamp is taken.

In the second, non-critical part, the transmission timestamp is transmitted to the recipient.

The following delays are taken into account for this process:

- **Send delay** $\Delta_{\text{send}}$
  
  This delay describes the time from the successful transmission of the message to the recording of the timestamp. It depends very much on which part of the system the timestamp is taken. Runtimes in operating systems, communication controllers, or task scheduling can play a major role here.

- **Network delay** $\Delta_{\text{network}}$
  
  This delay describes the time a message takes to be sent from the sender to the receiver. It consists of the transmission delay and the propagation delay. The transmission delay describes the time needed to get a packet on the line and is directly dependent on the baud rate. The propagation delay describes the time the packet spends on the line and is directly dependent on the line length.

- **Receive delay** $\Delta_{\text{receive}}$
  
  The receive delay describes the time from the moment it is present on the line until the timestamp is taken. The causes of this delay are similar to the send delay.

With the specified delay, the synchronization condition can now be specified more precisely. For the convergence function, the individual delay terms are now used, and the new precision condition is defined as
\[ \Delta_{\text{send}} + \Delta_{\text{network}} + \Delta_{\text{receive}} + 2\rho R \leq \Pi. \] (2.12)

This equation will now serve as a basis for various techniques to improve clock synchronization. From the synchronization condition it is clear that the precision can basically be improved in two ways. Either by reducing the convergence interval or by reducing the drift offset. The convergence interval can be reduced by decreasing the delays. One technique for this is to improve the timestamps.

### 2.3.1 Timestamping

There are two fundamental techniques for timestamping [RFSS17], [MES14], software timestamping and hardware timestamping. They differ essentially in the part of the system where the timestamp is taken. In the case of hardware timestamping, this takes place in the communications controller or in dedicated hardware. In the case of software timestamping, the clock is queried by the software at certain times.

In the equation 2.12 the terms \( \Delta_{\text{send}} \) and \( \Delta_{\text{receive}} \) are directly influenced by this. In general, it can be said that the closer the timestamp is to the communication medium, the more accurate it is. This leads to shorter transmission and reception delays.

Software timestamps are sometimes subject to very large fluctuations, since the time point is also determined by the application process. Waiting times in the operating system can lead to gross inaccuracies of timestamps. In the worst case, messages may also have to pass through the entire protocol stack to be timestamped. The great advantage of this method is its simplicity, since no additional or special hardware is required. In some cases special hardware is available, but it is difficult to access it. One reason for this can be special architectures like virtual machines.

Hardware timestamps can be taken independently of the software [MS15]. This means that they can be recorded with a constant small delay. They are usually recorded or triggered by the communication controller itself. It is also important at what point in time the timestamping occurs, this can be at the beginning, end or special parts of a packet. A common scenario is for the communications controller to integrate its own high-precision clocks. The disadvantage of this solution is that dedicated hardware is required and the associated higher costs.

### 2.3.2 Clock adjustment techniques

To improve the drift offset, different possibilities of clock adjustment are used. In principle, clocks can be adjusted in three ways.

- **Offset adjustment**
  
  This is the simplest way of adjusting the clock. Here the clock is adjusted directly by jumps. This can be done directly by either decrementing or incrementing the entire counter. One possibility of indirect offset correction is to write the resulting error into a separate register.

- **Rate adjustment**
  
  In this variant, the resulting error is compensated by changing the rate.


- **Offset and rate adjustment**

  This variant uses both variants and adjusts both by jumps and by the rate.

  As already mentioned, offset adjustment is the simplest variant. The advantage is that the error is instantly corrected. The disadvantage, especially with the direct adjustment, is that the local clock does not provide unambiguous times, because one and the same timestamp can occur more often. This causes problems especially with the unique time allocation of events or measurement data. The method of indirect offset adjustment partially solves this problem, since the local clock is always counted monotonously. However, problems can still occur with the absolute time, i.e. the time of the counter added to the offset. Examples of possible errors are the turn of the year, which may be counted twice, or leap seconds that are used twice. These problems are completely solved by rate adjustment. Each point in time is unique as only the speed of the clock is affected. For the type of rate adjustment, knowledge from control engineering is often used. The use of regulators is intended to prevent the clock from becoming unstable or from over speeding. The disadvantage of a pure rate adjustment is that it takes a very long time to become synchronous in case of large errors. This disadvantage can be compensated by a combined solution. This is often handled in such a way that an offset adjustment is carried out after a certain error. This is the case, for example, when starting the synchronization.

  For the drift offset, the three methods have a different influence. Pure offset adjustment brings the clock back to the convergence interval. Therefore the drift offset can only be improved by reducing the sync interval. However, this results in an increased bandwidth usage. With a rate adjustment the drift rate can be influenced directly. This allows a better precision to be achieved with the same sync interval as with pure offset adjustment.

**2.3.3 Delay measurement**

The network delay cannot be influenced directly, since it is largely determined by the cable length. However, there are various ways to determine the delay or to minimize the influence. The simplest variant is to assume a minimum delay for the data transmission. This can be calculated from a minimum line length and a known transmission delay at a known data rate.

Delay measurements offer a much more accurate way to minimize the influence of line delay [RFSS17]. This can be done in different ways. A very common variant is to measure the packet round trip time. This is done by sending a measurement packet. This is sent and immediately returned. A send and receive timestamp is taken. Figure 2.6 shows the time course of the packet. The packet is sent at time $t_1$ to the responder and received at time $t_2$. Then sent back to the initiator at time $t_3$ and received at $t_4$. If the timestamp $t_4$ cannot be inserted directly to the message, a separate message is needed. At the end of this process, the delay initiator knows the timestamps $t_1$, $t_2$, $t_3$ and $t_4$.

The round trip time $t_{RTT}$ can now be calculated as

$$t_{RTT} = t_2 - t_1 + t_4 - t_3.$$  \hspace{1cm} (2.13)

Assuming a balanced connection, the network delay can then be calculated according to

$$\Delta_{Network} = \frac{t_{RTT}}{2}.$$  \hspace{1cm} (2.14)
This variant is very well suited for balanced links [EBS14]. However, the accuracy of the measurement is highly dependent on the quality of the timestamp.

2.4 Controller Area Network

The CAN [Gmb91] bus was originally introduced by Bosch to drastically reduce the amount of cabling in automobiles. Today it is the most widely used transmission standard in the automotive sector. The reasons for this are the simple structure and the low cabling effort as already mentioned. These characteristics have led to the CAN bus being used in other areas such as automation technology. Since its introduction, it has been continuously developed to meet new requirements. The current version is CAN FD, which will be extended in the future by the CAN XL standard.

This chapter is intended to provide an overview of the most important features of the CAN bus.

2.4.1 Basic concepts

The CAN bus is a serial bus system. It uses a line topology in which all stations are connected to the same bus. Bus access is via CSMA/CR. This procedure is not destructive, so a message is always successfully transmitted on the bus. This is realized by the two logic levels dominant and recessive. A dominant level always prevails over a recessive level. If one station sends a dominant level and one station sends a recessive level, only the dominant level appears on the bus. In the case of copper lines, the bits are transmitted differentially, but other transmission media such as optical fibers can also be used. The standard CAN bus has a maximum transmission rate of 1 Mbits. The maximum line length is defined by the data rate used and is 40m for the
maximum rate. The reason for this is that the maximum signal circulation times must be taken into account. These consist of the transceiver delays and the signal propagation time. This is especially important because all messages must be received and acknowledged by one. If the line were too long, the acknowledgement would take too long and there would no longer be a common consistent view of the bus. The maximum participants per network depend on the transceiver used.

The arbitration of CAN messages is done via message identifiers, whereby the lowest identifier always prevails. A further feature is the bit stuffing. The CAN bus allows bit sequences with more than five identical bits only in control signals. In fields where this does not apply, a stuffing bit is inserted after five equal bits. This bit has the inverted level of the previous bit sequence. The fields Acknowledge, CRC and End of frame are excluded from bit stuffing.

A standard CAN message can transmit a maximum of eight data bits.

2.4.2 Messages

Four different message types are defined for the CAN bus: Data frames, Remote frames, Error frame and overload frame. Data frames are used to transmit user data. A remote frame is used to request data from another CAN node. An error frame is transmitted when an error is detected on the bus and an overload frame is used to insert an extra delay between two frames. Here we only deal with the data frame, because it is the only one used for data transmission and this is the core task of a transmission system.

A data frame consists of seven fields: start of frame, arbitration field, control field, data field, crc field, acknowledge field and end of frame. Figure ccc shows the structure of the standard data frame.

| Arbibration | Control | Data | CRC | Ack | EOF |

- **Start of frame**
  It consists of a single dominant bit and marks the beginning of a data transmission. Before that, the bus is in idle state and therefore has the value of a recursive bit. At this point it is still possible for several transmitters to transmit simultaneously.

- **Arbitration field**
  The arbitration field consists of the identifier and the remote transmission request bit. The identifier consists of 11 bits in standard format and 29 bits in extended format. In this phase of the transmission it is decided which station gets access to the bus. The station with the least significant identifier always wins. If several participants send their identifier at the same time and one of them detects a dominant level although it is sending a resume, this participant stops and the other one wins the arbitration. The rtr bit is always dominant in the data frame. If the extended format is used, a substitute remote request and an identifier extension bit is inserted after 11 identifier bits. This is for backward compatibility.
• **Control field**
  The control field consists of six bits. The structure differs depending on whether it is a standard frame or an extended frame. In a standard frame, the first bit is the ide bit, which is already contained in the arbitration field of an extended frame. In an extended frame, the first bit is reserved for future protocol extensions, this also applies to the second bit regardless of the message version. The last four bits contain the data length code, which determines the length of the data field.

• **Data field**
  The data field has a variable length, which can vary from 0 to 8 bytes. This is determined by the data length code in the control field. The content of the data field consists of payload.

• **CRC field**
  The CRC field consists of the CRC checksum and the CRC delimiter and has a size of 16 bits. The CRC checksum is 15 bits, followed by the recessive delimiter.

• **Acknowledge field**
  The Acknowledge field is used to confirm the data transfer. It consists of two bits, an acknowledge slot, and a delimiter. In the acknowledge slot, a recessive bit is transmitted by the sender of the message. All nodes that have received the message correctly overwrite this with a dominant bit. If the level remains recessive, the sender recognizes that no other node has received the message and tries again. The acknowledge delimiter consists of one recessive bit, this ensures that the acknowledgment of the other nodes in the ack field is always surrounded by two recessive bits.

• **End of frame**
  The end of frame field terminates the CAN data frame and consists of seven recessive bits.

### 2.4.3 Bit timing

So that all stations have a consistent common view of the bus, the length of a bit must be the same for all stations. For this purpose, CAN has a bit synchronization, which determines the sampling time for individual bits.

In CAN, individual bits are multiples of so-called time quanta. Time quanta are derived from a local oscillator by a prescaler. The values of this prescaler are between 1 and 32. Time quanta are always integer multiples of the minimum time quantum. The number per bit is between 8 and 25.

A single bit is divided into four segments on the CAN bus.

• **Synchronization segment**
  This segment is to synchronize all nodes. A rising or falling edge should lie within this segment. It has a fixed length of one time quantum.

• **Propagation segment**
  This segment is used to compensate for signal delay and other delays. It has a length from 1 to 8 time quanta.
State of the art

- Phase buffer segment 1
- Phase buffer segment 2

The two segments have a variable length and are used to vary the sampling time.

The sampling point is always between phase buffer segment 1 and phase buffer segment 2.

When changing from idle bus state to a dominant level, as it occurs at start of frame bit, the bit timer is restarted. This is called hard synchronization and typically occurs at the beginning of a frame.

For edges within a frame, it is detected in which segment the edge occurs. If the edge is in the synchronization segment, there is no error and there is no need to synchronize.

If the edge is before the synchronization segment, there is a negative phase error, if it is after the synchronization segment, the error is positive. The phase error is always measured in time quanta.

Such a phase error is corrected by resynchronization. If the phase error is positive, the phase buffer segment 1 is extended. With a negative phase error the phase buffer segment 2 is shortened. The lengthening or shortening is determined by the phase error. The maximum is given by the maximum resynchronization step width.

2.4.4 CAN FD

The CAN bus is still under development, the next stage after the classic CAN bus is CAN with flexible data rate [Bos11]. This standard increases the maximum transmission rate to 4 Mbits. In addition, 64 bytes instead of 8 bytes can be transmitted in one message. This is made possible by bit rate switching. Two different bit rates are used in this procedure. The start of frame bit, arbitration and parts of the control field are transmitted with the standard CAN bit rate. A bit in the control field then indicates that bit rate switching is activated. From then on the bit rate is increased. After the CRC field the bit rate is changed back to the standard CAN bit rate. With the opened bit rate, 64 bytes can be transmitted in the same period of time as with classic CAN 8 bytes can be transmitted. Another change is that an improved CRC algorithm is used. This requires a larger CRC field.

To make the increased data rate possible, extensions in the bit synchronization are necessary. This requires a separate set of configuration parameters for each data rate used. Therefore, the division of the time quanta is different for both data rates. The same applies to the synchronization behavior. To detect errors on the line more quickly, a transceiver delay compensation method is also used. In this method, the time from sending the bit to the signal change on the line is measured and the sample point is shifted by this time.

2.4.5 CAN XL

The third generation of CAN is called CAN XL [Aut20] and is currently under development. It promises data rates up to 10 Mbits and message sizes up to 2048 bytes. This should make it possible to use the CAN bus as a basis for the Internet Protocol. Furthermore the standard should be compatible to CAN FD.
2.5 Precision time protocol

The precision time protocol [IEE20] is a very common solution to distribute time in networks. It is used where a high precision is needed. Typical fields of application are distributed measurement systems or the field of automation.

The protocol is defined in the application layer of the TCP/IP protocol stack. It is not dependent on any physical layer or transport protocol, but it is typically used in IP networks.

2.5.1 Basic concepts

The basic concept of PTP is to distribute high precision time in computer networks. It is a hierarchic protocol with time masters and slaves. The mechanism works completely distributed, which means in this case, that every slave finds its master and synchronizes itself to it. A PTP network allows three types of network nodes, called clocks.

- Ordinary clock
  Ordinary clock can act as master or slave in the PTP network, but not both. They are synchronized directly to their synchronization partner.

- Boundary Clock
  Boundary clocks are used to transport time information across network boundaries. They typically have several ports, with which they take on different roles. For example, the time can be obtained from one network where the port acts as a slave and distributed to another where the port acts as a master.

- Transparent Clock
  Transparent clocks are not directly recognized as participants in PTP networks. They are only used to improve the accuracy. They do this by accepting PTP messages, inserting correction information into the packet and forwarding them. This role is typically taken over by PTP capable switches.

2.5.2 Time distribution mechanism

In PTP networks, time is distributed in two steps. First, the absolute time is transmitted using SYNC messages. Then the delay is determined by measurement. The PTP protocol has two types of delay measurements: End-to-end and peer-to-peer.

The whole process is shown in figure 2.8.

First a SYNC message is sent by the master, this is timestamped by the master when sending and by the slave when receiving. The master now has the transmit timestamp t1 and the slave the receive timestamp t2.

Next a FOLLOW_UP message is sent. This is not timestamped. It contains the timestamp t1 in its user data. Thus the slave now has the timestamps t1 and t2. Thus the slave already knows the time of the master. However, this time is still falsified by the transmit delay.
This process is repeated periodically. If a specialized hardware is used, the FOLLOW_UP message can be omitted, because it is possible to insert the timestamp \( t_1 \) "on-the-fly" into the SYNC message.

Next, the slave sends a DELAY_REQUEST message. This message is timestamped by the slave when it is sent and by the master when it is received. At this time the slave knows the send timestamp \( t_3 \) and the master knows the receive timestamp \( t_4 \).

This is followed by a DELAY_RESPONSE message, which is also timestamped during transmission and reception. The message also contains the timestamp \( t_4 \). The slave now knows the timestamps \( t_3, t_4 \) and the receive timestamp of the DELAY_RESPONSE message \( t_6 \). The transmit timestamp \( t_5 \) is sent either via a DELAY_RESPONSE_FOLLOW_UP message or directly with the DELAY_RESPONSE message, depending on the hardware used.

The described procedure for delay measurement is peer-to-peer. The delay measurement is performed periodically on all available ports. The end-to-end method differs in the use of timestamps. Here, \( t_5 \) and \( t_6 \) are not measured and \( t_1 \) and \( t_2 \) are used instead.

The offset \( t_{offset} \) between master and slave can now be calculated according to

\[
t_{offset} = t_1 - t_2 - \frac{(t_4 - t_3) + (t_6 - t_5)}{2}.
\]  

(2.15)

It is assumed that the delay is symmetrical and therefore the same in both directions.
2.5.3 Best master clock algorithm

The BMCA is used in PTP networks to assign roles to each port and to find the best master in the network. A grandmaster is determined, this is the absolute time reference for the entire network. The selection is made by a number of criteria such as adjustable priorities, clock type, accuracy and variance of the clock. Each clock determines itself which clock should be the Grandmaster. The own clock is also taken into account in the selection. A Grandmaster can also lose its role. For example by adding a more accurate master or by losing a GPS signal. The quality of the clocks in the network is announced via ANNOUNCE messages. Each Grandmaster sends them with a certain interval. The BMCA also controls which role a port plays, i.e. whether it is master, slave or passive. The network topology also plays a role here. If a grandmaster can be reached via several paths, the shortest one is chosen. This is the path that can be reached via the fewest boundary clocks.

2.5.4 Specifics

This chapter describes only the most important functions of PTP. The entire standard offers much more than just specification on different network layers and management functions for synchronization networks. An important standard that specifies the use of PTP over Ethernet is IEEE802.1AS[IEE11]. This forms the basis for AUTOSAR time synchronization over ethernet[ASE17]. This standard is used in the automotive industry and forms a very simplified subset that is adapted to the requirements of networks in cars. For example, no BMCA is used because the roles and networks are predefined and static. Furthermore, only peer-to-peer delay measurement is supported. The standard is limited to the messages: SYNC, FOLLOW_UP, DELAY_REQUEST, DELAY_RESPONSE and DELAY_RESPONSE_FOLLOW_UP.

In general, the PTP standard describes how a time can be distributed in a network with very high precision. However, it does not specify how the clock should be set, i.e. what form of correction should be applied. This has the consequence that time jumps can occur with pure state correction. Common implementations like linuxptp, however, rely on a combination of rates and offset correction.

2.6 Autosar time synchronization over CAN

Autosar time synchronization over Can[ASC17] 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. The message format and the process is specified. Compared to PTP, it is a very weak 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.

Autosar time synchronization over CAN is a master-slave protocol. The roles are distributed by predefined configuration and are therefore static. The message transport is only from master to slave and is therefore purely unidirectional. The standard defines a timestamping of messages by sending and receiving acknowledgements, which are done purely via software. The time is corrected purely by state correction.

The time is transmitted via two messages, SYNC and FUP. The offset is optionally transmitted via OFS and OFNS messages.
Figure 2.9 shows the schematic transmission sequence. The function \( s(t) \) returns the second part of the timestamp. The function \( ns(t) \) returns the nanosecond part of the timestamp.

First the timestamp \( t_1 \) is taken, this is done by reading the clock. Only the second part of this timestamp is used.

The next step is to send the sync message. This contains the second part of \( t_1 \). If the message is sent successfully, the timestamp \( t_2 \) is taken from the master. At the slave the timestamp \( t_3 \) is taken at reception.

Then the time between \( t_1 \) and \( t_2 \) is calculated and sent with the time of the FUP message. For the calculation of the difference the already sent second portion of \( t_1 \) is subtracted from \( t_2 \). The FUP message is not timestamped.

The last step is the transmission of the offset, this is optional. Therefore the second part of the offset is transmitted in the OFS message and the nanosecond part of the offset is transmitted in the OFNS message. These messages are not timestamped.

### 2.7 Related work

In this section, the scientific state of the art on the topic of clock synchronization via CAN is analyzed and discussed for applicability to this work.
2.7.1 Approaches with time triggered communication

The authors of [DB16], [DBM17] study a basic solution to a synchronization problem. The main purpose of their work is not the synchronization algorithm itself, but the optimization of the bus utilization. The main problem they tackle is contentions on bus between multiple nodes. Nevertheless a very simple method to synchronize a clock is presented in this paper. They introduced a timing scheme with major and minor messages. The transmission of a major message is taken as reference for the timing of the remaining nodes. Communication only happens in a time triggered way. This solution works with software timestamps, therefore the delay is hard to determine, but they defined upper and lower limits based on worst case times. They achieve an accuracy of 1 millisecond, which is enough to tackle their problem. With regard to this work, this solution can contribute to the basic understanding of timing in CAN. However, the presented synchronization algorithm is completely unsuitable because of the time triggered communication scheme and the precision.

Key points:
- Main focus on bus utilization
- Theoretic precision of 1 millisecond

The main focus of [CP09] is the implementation of time division multiple access on the CAN bus. The clock synchronization method is very simple, because it uses properties of TDMA to get synchronized. The main part of this work is the delay estimation of the messages, which can be useful for this work. The achieved precision is less than 1 microseconds. This is mainly achieved by the properties of TDMA and a high resynchronization rate. A disadvantage with regard to this work is that no event triggered communication is possible.

Key points:
- Implementation of TDMA scheme
- Theoretic precision of 1 microsecond
- High bus load due high synchronization rate

The main goal [BKSG17] is to emulate multiple CAN nodes on a multicore processor. It is tried to split a physical CAN port as efficiently as possible to several applications. The second major point in this paper is the clock synchronization of the different units. Autosar time synchronization over CAN is implemented directly according to the specification. This is of particular interest for this work, because this mechanism is also to be used. In this case an accuracy in a range of 4.96 to 6.56 microseconds is achieved.

Key points:
- System on chip network
- Implementation of Autosar time synchronization over CAN
State of the art

- Precision in range of 4.96 to 6.56 microseconds

A different approach is investigated in [BSG17]. It offers a very sophisticated solution for clock synchronization on the CAN bus. The mechanism is implemented on an MPSOC partly on hardware and partly on software. It is therefore not a physical CAN network, but an on-chip network. The synchronization protocol is based on Autosar time synchronization over CAN. However, it is used in a modified form. Due to a change in the physical layer of the CAN controller, only one instead of two messages are needed for synchronization. Furthermore, information about bit timing is used to increase the accuracy. The message transmission is done in a TDMA scheme, this also contributes to an improvement, because properties of it are used. For this work the use of bit timing is especially interesting, because it does not require any changes to the hardware. However, the presented solution is limited to classic CAN and cannot be directly adapted to CAN FD.

Key points:
- System on chip network
- Modified Autosar time synchronization over CAN
- Precision of 2 microseconds

TTCAN is an extension of the CAN standard for time-controlled communication [FMD+01], [HMF+02]. The standard is divided into two expansion stages. The first stage only allows time-controlled communication triggered by a master time. An intervention in the hardware of the controller is not necessary. However, no global time base is transmitted. TTCAN level 2 extends the standard by a global time base. For this purpose a separate time unit called Network time unit is introduced. This time unit is derived from the local oscillator and synchronized with the network time. The second level is based on hardware timestamping, therefore a special controller is required for its use. The procedure for this starts at the lowest level of the protocol stack and is also used in [RnBP03]. The time synchronization achieves a precision of 10 microseconds [RNRP08].

Key points:
- Specialized CAN Controller
- Precision of 10 microseconds

2.7.2 Approaches with event based communication

The study in [MVLF08] examines the use of low cost hardware in network control systems. It uses precision time protocol, which is normally applied to ethernet communication. Due to the restrictions of CAN, mainly the payload length, the protocol was adapted. The timestamping is completely software-based. Nevertheless, it is an interesting approach, because they use simple hardware and software architecture. They achieve a precision of 1 millisecond, which is limited by the clock resolution. The proposed solution is interesting, because the message format is similar to Autosar.
Key points:

- Low cost implementation of PTP over CAN
- Precision of 1 millisecond

The approach of [GS94] provides a basis for precise clock synchronization on the CAN bus. It represents a basis for many subsequent works and mechanisms. In Autosar time synchronization over CAN the described principle is directly implemented. The variation of the local clock is given in a range of 20 microseconds. This work treats the algorithm only theoretically and does not provide an evaluation of its precision. However, the approach is taken up by other works that include measurements of the precision. Therefore, the evaluation from [AL04a] is seen as a reference for this algorithm. There a precision of $\sim 4\mu s$ is achieved. The Fundamental Findings can be seen as a starting point for this work, since the basic principles are the same. However, the hardware design differs immensely.

Key points:

- Basic work for clock synchronization over CAN
- Precision of 4 microseconds

In [ASS19], the properties of the CAN bus will be discussed in more detail, specifically with respect to Precise clock synchronization. The first is a pure software algorithm based on [GS94]. This offers a pure state correction. The potential of this method depends on the bus load. The second algorithm aims at rate correction and works with the bit timing of CAN. It uses the phase error detected by the controller. The first algorithm achieves an average precision of about 2 microseconds, depending on the bus load. Combined with the second algorithm, a precision of 1.1 microseconds is achieved. These results refer to results of a simulation. However, they were not evaluated on real hardware. No changes were made to the CAN hardware and no additional hardware was implemented, but it was assumed that the used CAN controller provides access to the current status of the bit timing. This assumption limits the selection of the CAN controller. Therefore there is still potential in the area of timestamping.

Key points:

- Approach that uses properties of CAN
- Assumes that the internal state of the controller is known
- Simulated precision of 1.1 microseconds

The main focus of [RGR98] is fault tolerance. The presented mechanism moves away from the master-slave paradigm and instead tries to create an algorithm for distributed systems. A voting system for the main clock will be introduced. The advantage of this is that there is no longer a single point of failure. The basic algorithm for clock synchronization is based on [GS94]. However, the clocks are only adjusted locally, although it is mentioned in the end of the paper that this can easily be changed. The results of the thesis focus very much on bandwidth usage of the CAN network. Only maximum values are given for precision.
State of the art

Key points:
- Main focus on fault tolerance and bandwidth usage
- No evaluation of precision

In [RnBP03] a complete clock subsystem for the CAN bus is presented. The focus is on precision and fault tolerance. Furthermore it is taken care that the system can be implemented orthogonally, which means that it can be used parallel to the standard CAN implementation without changing it. The solution should support common paradigms of distributed real-time systems, such as event and time-triggered communication. An additional CAN transceiver is required for the presented solution. This is connected to the presented clock unit and operates completely independent of the actual CAN controller. The synchronization algorithm is strongly based on TTCAN, since here too timestamps are written directly when the message is sent. A major enhancement to TTCAN is that this solution takes more errors into account. In the mentioned thesis only the mechanism and the architecture is presented. An evaluation will be presented in further papers.

The authors of [RNPH05] work on a continuation of the work of the clocks subsystem. Here the algorithm presented in [RnBP03] is formally checked. This is done using timed automata in the model checker UPPAAL. The model contains the communication via CAN and the introduced synchronization algorithm. Furthermore, a variable clock drift is modelled.

The work discussed in [RNRP08] provides an evaluation of the clock subsystem. It summarizes the results of the idea and the formal verification. Furthermore, the solution is compared to other common solutions for clock synchronization over CAN. Like in the basic idea, there is a strong focus on fault tolerance and precision. The final evaluation shows, that a precision of 20 microseconds is achieved. One interesting aspect of this work is, that clock amortization is used instead of timer jumps.

Key points:
- Requires additional CAN Controller
- Strong focus on fault tolerance
- Precision of 20 microseconds

The work of [Har17a] provides an implementation of AUTOSAR time synchronization over CAN with hardware timestamping. It uses the solution of [Har17b]. Timestamping happens separately from the controller, but is not completely independent of it, since a special controller is required. The main difference to TTCAN [FMD+01] [HMF+02] is, that the timestamps are taken at the end of frame instead of the beginning. This is necessary for the Autosar implementation to work together with software based solutions.

Key points:
- Implementation of Autosar time synchronization over CAN
- Requires specific CAN Controller
- No evaluation of precision

22
2.8 Conclusion

In contrast to some of the works presented, the focus here is purely on the distribution of time. It is about using a priori interfaces to distribute time in a distributed measurement and simulation system. Some of the presented works ([DB16], [DBM17], [CP09], [BSG17], [FMD+01], [HMF+02]) use a time controlled transmission scheme to increase bandwidth usage and accuracy. This is explicitly not desired here, because the communication should be event triggered only. In [RnBP03] different concepts are presented to increase the reliability, this is also not needed because the main focus is on synchronization accuracy and the assumption that there is only one master at a time. This assumption can be made because of the required accuracy, since only one device has a high precision GPS clock. Most of the presented mechanisms use a own message format. This is not relevant for this thesis, because the system is based on AUTOSAR time synchronization over CAN, which is based on the idea of [GS94].

The analysis of similar work leads to the conclusion that the problem definition of this work cannot be completely fulfilled by any previous work. A promising approach is offered by [Har17a], which extends the software-based AUTOSAR time synchronization over CAN by hardware timestamping. However, this approach does not address the properties of CAN, as is the case in the work of [BSG17] and [RnBP03]. Furthermore, no information is given on the accuracy and precision achieved. The [BSG17] solution also uses the AUTOSAR standard. However, this is modified so that only one message is needed. This leads to a strong interference in the hardware of the CAN controller and to an incompatibility to standard compliant devices. The precision is about two microseconds. However, it is important to remember that this is an implementation on a SOC and not on a physical CAN bus. The work of [RnBP03] offers a very sophisticated approach, because the properties of the CAN bus are used very well. Furthermore, the clock amortization is discussed, which will be dealt with in more detail in this thesis.

Clock correction methods

The way the rate and offset is corrected is explicitly treated in two papers. TTCAN corrects the rate based on the last synchronization interval, in which a quotient is formed from the interval of local and global time [FMD+01]. This quotient called time unit ratio, is applied directly to the rate each synchronization round. The offset is corrected by a separate register, this is added to the local time. The value of the register is calculated each round from the difference between global and local time and written to the register. Another approach is the pure rate correction [RNR08]. Here the offset is calculated each synchronization round and used as a basis for the clock rate. Here the new rate is calculated to compensate for the calculated offset within one synchronization period.

Table 2.1 provides an overview of the most promising methods. Only works that include a practical realization and evaluation were selected.

The solutions in this work are developed exclusively under the aspect of calculating the highest possible accuracy and precision. Full compatibility with Autosar is a key requirement. This applies to the message format as well as to the time of timestamping. Furthermore, the rates and offset correction should be independent of each other. The accuracy of the rate correction should be improved by using the properties of the CAN bus and suitable filtering. The offset correction is to be improved primarily by hardware timestamping, whereby clock amortization plays a major role here. For evaluation purposes the solutions will be compared to a purely software-based solution.
### State of the art

#### Relevant solutions in comparison

<table>
<thead>
<tr>
<th>Solution</th>
<th>Precision</th>
<th>Communication scheme</th>
<th>Changes in CAN controller</th>
<th>Timestamping mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTCAN level 2 [FMD+01]</td>
<td>$\sim 10,\mu s$</td>
<td>TDMA</td>
<td>Yes</td>
<td>Hardware</td>
</tr>
<tr>
<td>Lee et. al. [AL04a]</td>
<td>$\sim 4,\mu s$</td>
<td>Event-triggered</td>
<td>No</td>
<td>Interrupt based</td>
</tr>
<tr>
<td>Breaban et. al. [BSG17]</td>
<td>$\sim 2,\mu s$</td>
<td>TDMA</td>
<td>Yes</td>
<td>Hardware</td>
</tr>
<tr>
<td>Rodriguez-Navas et. al. [RnBP03]</td>
<td>$\sim 20,\mu s$</td>
<td>Event-triggered</td>
<td>Yes</td>
<td>Hardware</td>
</tr>
</tbody>
</table>

Table 2.1: Relevant solutions in comparison
3 System overview

This chapter gives an overview of the platform on which this work is implemented. The different components that are important for clock synchronization or precision evaluation are described. At the end of the chapter a delay analysis is performed to determine the delays in the timestamping process.

The solution is implemented on the EBX200 platform. It is a modular multipurpose embedded system. The system consists of a base board that can be equipped with different bus modules.

The main field of application is simulation and measurement tasks in the automotive sector. Especially for capture-replay tasks a highly accurate time synchronization is required.

3.1 Device structure

The EBx200 baseboard forms the core of the platform. The main components are a dual core PowerPC processor, an Altera Cyclone V FPGA and a clock. This base board can be used with various XMC carriers. These provide the power supply and the communication interface to the host PC. Depending on the selected host interface, the EBX200 is a standalone device or a PCIe interface board. As a standalone device, it can be connected to a host computer via TCP or USB.

Depending on the design, up to eight ports are available for bus modules.

The processor on the EBX200 platform contains two cores, each of which is used for different tasks. One core runs a linux operating system and handles the non-time-critical tasks such as connecting to and transferring data to the host PC. The second core is used to perform real-time tasks, including clock synchronization.

3.2 Hardware part

This section introduces the relevant functions for time synchronization that are implemented in hardware. These are implemented as self-contained units in the FPGA. Except for the Bosch MCAN Controller all mentioned modules are proprietary developments of Elektrobit.
3.2.1 Bosch MCAN

The MCAN [Bos15] from Bosch is used as CAN controller. It is integrated as IP core in the FPGA and connected to the physical bus via a transceiver. The controller supports both standard CAN (2.4) and CAN FD (2.4.4).

The analysis of the controller is important for this work in view of the extension of the clock synchronization functionality. In the field of Ethernet based transmission it is already established that communication controllers offer built-in hardware for hardware timestamps. This is one reason for the wide use of PTP (2.5). There the timestamp is typically generated directly between the physical and the media control layer.

The MCAN Controller offers an integrated function for recording timestamps. Transmission and reception timestamps can be generated. The function can be assigned to the data link layer. The timestamps are recorded at the beginning of the message. This circumstance leads to the fact that the function cannot be considered for this work, because an end timestamp is necessary for Autosar time synchronization over CAN.

Figure 3.2 illustrates where the timestamp is taken in the protocol stack. The arrow indicates the point in the protocol stack where the timestamp is taken. While in PTP a timestamp can typically be taken directly between the physical and data link layer, in the MCAN controller it is taken after bus arbitration which leads to additional delays.

Since the controller also supports CAN FD a transceiver delay compensation is used. This can also be used to improve the timestamp accuracy. With this mechanism the delay caused by the transceiver is measured and used to determine the sampling point. The measured delay can be
read out directly via a register interface. But for this it is necessary that a CANFD message has been sent. Otherwise no delay is measured. How and if this measurement is used is described in chapter 3.6.

In [RnBP03] the CAN bit error is used directly to correct the rate of the local clock. This procedure cannot be considered with the MCAN controller, because it does not provide an interface for the current correction value of the sampling point.

### 3.2.2 Frame detection unit

For the creation of timestamps, a separate IP core is used, which works independently of the CAN controller. This block is called frame detection unit and is directly connected to the CAN lines of the transceiver. Figure 3.3 shows the basic circuitry of the frame detection unit. The clock in the circuit diagram corresponds to the corrected clock from the EB clock synchronization IP core.

The frame detection unit reads the received signals on the bus. This is done via the receive interface of the CAN transceiver. There both sent and received messages are tracked. The bit time is determined via the set baud rate. The sampling point is also determined from the configuration of the MCAN Controller.

In principle the frame detection unit knows three states on the bus:

- **IDLE**
  
  In this state there is no data on the bus. The frame detection unit waits for an edge from recessive to dominant. If this edge occurs a start timestamp is generated. After that the state changes to FRAME.

- **FRAME**
  
  In this state a CAN frame is transmitted. The frame detection unit only counts the number of recessive bits. As soon as seven recessive bits occur in a row, a timestamp is generated. This happens because only the end of frame part of a CAN frame consists of seven consecutive recessive bits and thus breaks the rules of bit stuffing. So the end timestamp is generated. After that, it changes to the INTERFRAME SPACE state.
• INTERFRAME SPACE

In this state the interframe space is waited for. After three bits, the system automatically switches to the state IDLE.

The timestamp point of the frame detection unit is therefore between CAN transceiver and MCAN controller and thus between data link layer and physical layer in the OSI model. Figure 3.4 illustrates this fact. Thus a similar point is reached as with PTP.

The advantage over the timestamp function of the MCAN controller is that the frame detection unit works completely independently and can also generate end timestamps. Figure 3.5 illustrates the position in the frame where the timestamp is taken. Furthermore the timestamp point is closer to the physical bus.

A more detailed analysis of the timestamp point and delays is given in chapter 3.6.

3.2.3 EB clock synchronization ip core

The EB clock synchronization IP core forms the basis for all timing processes in the EBX200 platform. It creates all times and timestamps that are required for the correct sending and receiving of data. It is implemented directly in the FPGA. It is responsible for the following points:

1. Provision of corrected and uncorrected time bases

Two different counters are derived from a 40 MHz quartz. Both operate as zero-based nanosecond counters with an internal width of 90 bits, of which 64 bits are allocated to the nanoseconds and the remaining 26 bits to the decimal places. The uncorrected raw counter is incremented by 25 nanoseconds for each crystal tick. The rate can be adjusted at any time.
2. Provision of offset interfaces

   For the offset a 64 bit nanosecond register is available. This register is interpreted as a signed integer. The reason for this is that with internal synchronization with a very small epochal timestamp, jumps in the counter can be avoided.

3. Capturing of jumps in offset or timers

   Every abrupt change of the corrected counter or offset is timestamped and provided by a register interface.

4. Management of synchronization sources

   Registers are provided to allow the selection of synchronization sources. Basically two types of sources are distinguished, hardware sources and software source. The big difference is that in the first case the synchronization is done directly by the IP core and the rate and
offset cannot be changed by software.

5. Management of the synchronization state

A register is provided to control and signal the synchronization state. This is written directly from the IP core in case of hardware synchronization. It is also used to display the synchronization status in the EB timestamp packets.

6. Provision of hardware timestamps

For external sources like GPS there are separate timestamp registers available which can be triggered by external pulses. In case of GPS the PPS pulse is recorded.

7. Recording of hardware timestamps

Input and output data are timestamped by the IP core. This data is provided by the EBHSCR protocol.

8. Rate Correction

The clock synchronization IP core processes the ticks of the connected oscillator. At each tick 25 nanoseconds are added to the timer value. But this value is variable for the corrected timer. A register can be used to set the tick length. This is represented by a 32 bit value. This value is stored in a fixed point representation. 6 bits fall into the nanosecond part and the remaining 26 bits form the fractional part. By changing them, possible deviations of the oscillator frequency can be corrected.

9. Offset correction

The IP Core offers three variants for offset correction. In the first variant, the corrected timer is directly overwritten. The second variant uses an extra offset register that is written independently of the corrected timer. This ensures that there are no direct jumps in the timer. In the third variant the offset is corrected by the rate. Two configuration registers are provided for this. With one of them a temporary tick length can be set. The other one is used to set the number of ticks with which this temporary rate should be applied.

10. Synchronization via hardware syncline

The synchronization over a dedicated synchronization line is also performed by the clock synchronization IP core.

3.3 Software part

The main part of the synchronization is done in software. This includes receiving, processing and sending messages. Furthermore, the EB clock synchronization IP core is controlled by software.

3.3.1 Realtime core

On the EBX200 platform, all tasks required for measurement and simulation are executed on the real-time core. A real-time operating system runs there that processes the various tasks with the help of a scheduler. These include the polling of all bus modules for new data, the communication and forwarding of received data to the connected host and the execution of send requests. The
architecture is based on the fact that time-critical tasks such as timestamping are performed in the FPGA and thus without variable delay. The scheduler also works cooperatively, which means that running tasks are never interrupted.

CAN frames are also received by this scheduler. It checks with a period of 500 microseconds whether messages have been received from the MCAN controller. If such messages are received, it can take up to one task period until they are registered by the software. Since received data is timestamped by the hardware, this does not cause any problems for the timing, since the sequence of the data is given by the timestamp. However, if one wants to generate software timestamps, this leads to variable delays and thus to error-prone data. In chapter 4.4 this problem is dealt with.

Applications on the device are also run by the scheduler, including clock synchronization. If software timestamping is used, such tasks rely on being notified as soon as a message has been received. In this case, they are notified by the CAN receive task as soon as this is the case. This causes considerable delays. These and the resulting problems are explained in chapter 3.6.

3.4 Elektrobit highspeed capture and replay protocol

Since the EBX200 is also used for capture tasks, the data must be prepared in a suitable form and made available to the user. The elektrobit highspeed capture and replay protocol was introduced for this purpose. It extends received and transmitted data with a header that contains hardware-based timestamps and additional information about the data. This is relevant for this work because this header is also used as a source for the timestamp for synchronization tasks.

Figure 3.6 shows the structure of the header.

The header consists of the following fields.

- **Major number**
  
  This value indicates the type of data. These can come from different communication interfaces.

- **Slot**
  
  This value is only relevant when using several capture units. It indicates the type of data from which the data originates.

- **Channel**
  
  A communication module can have several channels. This value indicates the channel from which the data originates.

- **Version**
  
  This value indicates the protocol version. The presented version is version 0.

- **Status**
  
  This field is used to indicate the status of the data and possible errors. Each type of data has its own interpretation of the values.
System overview

• Payload length
  This field indicates the length of the payload excluding EBHSCR header.

• Start timestamp
  This field contains the 64 bit nanosecond timestamp of the beginning of the message. The exact time of the timestamp depends on the type of message.

• Stop timestamp
  This field contains the 64 bit nanosecond timestamp of the end of the message. The exact time of the timestamp depends on the type of message.

• Major number specific header
  This field contains a specific header for each type of message.

<table>
<thead>
<tr>
<th>Major number</th>
<th>Sl.</th>
<th>Channel</th>
<th>Version</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Payload length</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Start timestamp</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Stop timestamp</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Major number specific header</td>
</tr>
</tbody>
</table>

Figure 3.6: Elektrobit highspeed capture and replay protocol header

3.4.1 CAN frame

CAN frames have the major number 0x53 in the EBHSCR header. In this case the status field indicates whether the messages are CAN or CAN FD frames. The start timestamp is taken directly after the start-of-frame bit, the stop timestamp directly after the end-of-frame bit. In the major number specific header error codes and stats of the MCAN controller are written and the payload following the header directly maps the CAN message.

For this work it is very important that the stop timestamp is taken directly after the end-of-frame. This time is equivalent to the time of the send confirmation, which is required by AUTOSAR timesync over CAN. The EBHSCR header with its timestamps provides the basis for further work, because the hardware timestamp can be taken directly from it.

3.5 Relevant features for evaluation

The EBX200 platform offers features that are relevant for the evaluation and assessment of synchronization precision. These will be discussed in this chapter.
3.5.1 Digital Event Triggered Input

The EBX200 can also be used to capture digital signals. Pins are provided as inputs that react to rising or falling edges depending on the configuration. If an edge occurs at such a pin, a timestamp is generated in the FPGA. This timestamp is then made available to the software by a special measurement controller in the FPGA. This controller is regularly checked by the scheduler for new data and processed further depending on the application. In the simplest case, a data packet is generated from this and forwarded to the host. This feature is called digital event triggered input or DETI for short. Figure 3.7 shows how this function is mapped in the system.

This feature allows digital pulses to be timed. Potential delays in the generation of the timestamp, i.e. the time from the application of the voltage and the reading of the clock, can be regarded as constant, since they are completely implemented in the FPGA. This feature makes this function relevant for the evaluation of synchronization solutions. Because it allows the reading of the current time by digital pulses. If an edge is applied to the DETI ports of several devices at the same time, the current time can be read out simultaneously. This is used in chapter 5 to determine the synchronization precision.

3.5.2 Timer GPIO

Timer GPIOs are pins that are toggled directly by the internal clock. This function was not used to measure the synchronization accuracy between two devices directly. Potential delays between incrementing the clock and receiving the time can again be assumed to be constant, since the function is implemented directly in the FPGA. The pins reflect bits 31 to 26 of the corrected nanosecond counter. If the same pin is observed on several devices, they must always switch at the same time if the time is perfectly synchronous. Deviations between the edges can be interpreted directly as synchronization errors. Figure 3.7 shows how this part fits into the system.

3.6 Delay analysis

This chapter analyses the delays that occur when generating the timestamp. For delays, the term variable is used more often in the following chapter. In this context, this describes values that can change during operation. This also includes changes in cable length.

The analysis is based on a communication model shown in figure 3.8. The total delay is divided between different components. The following delays are used:

- $\Delta_{\text{Scheduler}}$
  
  This delay is caused by the realtime scheduler. It describes the time period 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. But under real conditions this delay is larger, because the receiving task can be delayed by other tasks. In addition, the value increases when several messages are received. Therefore this value can be assumed as highly variable.
Figure 3.7: Overview of timing related features

- $\Delta_{\text{read time}}$
  This part describes the delay in reading the time. It is the time between calling the read function and the actual reading of the timestamp. This value cannot be taken as constant because it depends on the execution time of the software.

- $\Delta_{\text{MCAN rx}}$
  This part describes the delay between a signal on the rx line of the controller and the time the message is available in the message memory. It can be assumed as nearly constant, because the CAN controller is completely implemented in hardware.

- $\Delta_{\text{transceiver rx}}$
  This part describes the delay between a signal on the CAN bus and a signal on the rx line of the transceiver. It behaves differently for rising and falling edges.

- $\Delta_{\text{transceiver tx}}$
  This part describes the delay between a signal on the tx line and a signal on the CAN bus. It behaves differently for rising and falling edges.

- $\Delta_{\text{propagation}}$

34
This part describes the delay between the signal on the CAN bus of the sending node and the signal on the CAN line of the receiving node. It is determined by the line length and is therefore variable.

- $\Delta_{FDU}$
  This part describes the delay between sampling point of the last EOF bit of the CAN message and writing the timestamp. It is nearly constant because the frame detection unit is completely hardware based.

Based on these delays, the delay for the different timestamp methods is now determined. The $t$ is taken as reference value. It describes the time of the last sampling point on the CAN bus seen from the senders perspective. The send delay is not important for the determination of the timestamp point, because sent messages are treated like received messages when determining the timestamp. The reason for this is the CAN bus. A message can only be considered as successfully sent when it is acknowledged and no errors occur. Since the CAN bus has a line topology, this can be determined by treating the sent message like a received message. The reason for this is that all participants have the same view of the bus and therefore also a common view of the frame and the errors.

### 3.6.1 Software timestamping

To determine the delays, the timestamps transmitted via AUTOSAR time synchronization over CAN are compared. This is the send timestamp on the synchronizations master $t_1$ and the receive timestamp $t_2$ on the synchronizations slave.

The figure shows the time sequence from the sending of the message to the generation of the timestamp. The time up to $t$, i.e. the duration until the message was successfully sent, is not
important in this case, since the message is received again by the sender. At the master the timestamp point is delayed by the reception of the message. The variable parts have the largest part of it. The timestamp $t_1$ point is thus calculated as

$$t_1 = t + \Delta_{\text{transceiver,rx}} + \Delta_{\text{MCAN,rx}} + \Delta_{\text{read time}} + \Delta_{\text{Scheduler}}. \quad (3.1)$$

At the slave the time is additionally delayed by the propagation delay. Timestamp $t_2$ is defined as

$$t_2 = t + \Delta_{\text{transceiver,rx}} + \Delta_{\text{MCAN,rx}} + \Delta_{\text{read time}} + \Delta_{\text{Scheduler}} + \Delta_{\text{propagation}}. \quad (3.2)$$

![Software timestamping delays](image)

**Figure 3.9:** Software timestamping delays

The synchronization algorithm assumes that the two timestamps $t_1$ and $t_2$ describe the same point in time. Therefore the total delay $\Delta_{\text{SWTS}}$ is the difference between the two timestamps and determined as
\[ \Delta_{SWTS} = t_2 - t_1 \\
= \Delta_{transceiver,rx,slave} - \Delta_{transceiver,rx,master} + \\
\Delta_{MCAN,rx,slave} - \Delta_{MCAN,rx,master} + \\
\Delta_{read,time,slave} - \Delta_{read,time,master} + \\
\Delta_{Scheduler,slave} - \Delta_{Scheduler,master} + \\
\Delta_{propagation}. \]

(3.3)

For constant delays we can assume that they are the same on both devices. In this case this concerns \( \Delta_{MCAN,rx} \) and \( \Delta_{transceiver,rx} \). This simplification can be made in this case because the differences of the delays are small compared to the variations of the scheduler. For the transceiver the assumption can be made, since both devices receive the same frame and thus receive the same edge. Therefore (3.3) can be simplified to

\[ \Delta_{SWTS} = t_2 - t_1 \\
= \Delta_{read,time,slave} - \Delta_{read,time,master} + \\
\Delta_{Scheduler,slave} - \Delta_{Scheduler,master} + \\
\Delta_{propagation}. \]

(3.4)

This makes it clear that the biggest influencing factors lie in the software. The variation of \( \Delta_{Scheduler} \) has the biggest influence and is therefore the determining part for the inaccuracy of the software timestamping.

Another problem is that the schedulers are not synchronized with each other. This circumstance and the variable delay caused by the scheduler lead to several problematic cases where large variations can occur. This problem becomes clearer if one look at the position of the schedulers to each other. One possible scenario is that the schedulers are synchronous to each other. This is shown in figure 3.10. The figures 3.11 and 3.12 show problematic cases.

![Figure 3.10: Coinciding scheduling periods](image)
Case 1

Figure 3.13 shows the best case scenario. Here master and slave timestamp are generated at the same time. The following assumption applies

$$\Delta_{Scheduler, master} + \Delta_{read\_time, master} = \Delta_{propagation} + \Delta_{read\_time, slave} + \Delta_{Scheduler, slave}. \quad (3.5)$$

From equation 3.6 follows then

$$\Delta_{SWTS} = 0. \quad (3.6)$$

If the two timestamps are now synchronized with each other, no error occurs between master and slave. The two time bases are therefore perfectly synchronous. However, since the schedulers cannot be specifically influenced, it is coincidental when this case occurs.
Figure 3.13: Case 1: Timestamps are created at the same time

Case 2

Figure 3.14 shows a borderline case. Here the error has a negative sign and reaches the minimum value. In this case $\Delta_{read\ time,\ master}$ is maximum and $\Delta_{read\ time,\ slave}$ is minimum. The scheduling delay $\Delta_{Scheduler\ time,\ slave}$ vanishes in this case, because packets are directly read at reception. The $\Delta_{Scheduler\ time,\ master}$ reaches its maximum. The propagation delay $\Delta_{propagation}$ is minimal here. These assumptions can be summarized as

\[
\begin{align*}
\Delta_{read\ time,\ master} &= \max \Delta_{read\ time} \\
\Delta_{read\ time,\ slave} &= \min \Delta_{read\ time} \\
\Delta_{Scheduler\ time,\ master} &= \max \Delta_{Scheduler\ time} \\
\Delta_{Scheduler\ time,\ slave} &= 0 \\
\Delta_{propagation} &= \min \Delta_{propagation}.
\end{align*}
\]  

From these assumptions then follows

\[
\Delta_{SWTS} = \min \Delta_{read\ time} - \max \Delta_{read\ time} - \max \Delta_{Scheduler\ time} + \min \Delta_{propagation}.
\]  

If the two time bases are synchronized here on the basis of the timestamps, the slave runs ahead of the master by $\Delta_{SWTS}$.
Case 3

Figure 3.15 again shows a borderline case. In this case the error has a positive sign and reaches the maximum value. It is assumed here that $\Delta_{\text{read time, master}}$ is minimum and $\Delta_{\text{read time, slave}}$ is maximum. The scheduling delay $\Delta_{\text{Scheduler time, master}}$ vanishes in this case and $\Delta_{\text{Scheduler time, slave}}$ reaches its maximum. The propagation delay $\Delta_{\text{propagation}}$ is maximum here. The assumptions can be summarized as

\begin{align*}
\Delta_{\text{read time, master}} &= \min \Delta_{\text{read time}} \\
\Delta_{\text{read time, slave}} &= \max \Delta_{\text{read time}} \\
\Delta_{\text{Scheduler time, master}} &= 0 \\
\Delta_{\text{Scheduler time, slave}} &= \max \Delta_{\text{Scheduler time}} \\
\Delta_{\text{propagation}} &= \max \Delta_{\text{propagation}}.
\end{align*}

(3.9)

From these assumptions then follows

\begin{align*}
\Delta_{\text{SWTS}} &= \max \Delta_{\text{read time}} - \min \Delta_{\text{read time}} + \max \Delta_{\text{Scheduler time}} + \min \Delta_{\text{propagation}}. \tag{3.10}
\end{align*}
In this case, the timestamp is generated by the master beforehand. If the two timestamps are now synchronized with each other, the master runs ahead of the slave.

The cases described show borderline cases. Of course, all cases in between and also a combination can occur. It is particularly problematic when switching between cases 2 and 3. This leads to the fact that time intervals can no longer be measured exactly. This strongly influences the rate correction.

### 3.6.2 Hardware timestamping

The procedure for hardware timestamping is similar to that for software timestamping. In this analysis it should be determined if the MCAN controller’s delay measurement adds value to the timestamping process.

First the delay of the variant without MCAN delay measurement is determined.

Figure 3.16 shows the time sequence of the timestamping. Here again it becomes clear why the sending time has no influence on the error. After the time \( t \) at which the message was successfully sent, a timestamp is generated at the master after the delays from the transceiver and the frame detection unit. At the slave the propagation delay is added.

The timestamp \( t_1 \) is now calculated as

\[ t_1 = \text{timestamp}_\text{master} - \text{sender delay} - \text{frame detection delay} - \text{propagation delay}. \]
For the total delay $\Delta_{SWTS}$ follows

$$\Delta_{HWTS} = t_2 - t_1 = \Delta_{\text{transceiver}_{rx}, \text{slave}} - \Delta_{\text{transceiver}_{rx}, \text{master}} + \Delta_{FDU, \text{slave}} - \Delta_{FDU, \text{master}} + \Delta_{\text{propagation}}.$$

(3.13)

A simplification like the software timestamping is deliberately omitted here, because the variations of the delays are in the order of magnitude of the total delay.

To find out if the delay measurement of the CAN controller can be used in a reasonable way, the same calculation is performed. The measured delay of the CAN controller is subtracted from the timestamp. The measured delay is a sum of $\Delta_{\text{transceiver}_{rx}}$ and $\Delta_{\text{transceiver}_{tx}}$. 

Figure 3.16: Delay of hardware timestamping without MCAN delay measurement

$t_1 = t + \Delta_{\text{transceiver}_{rx}} + \Delta_{FDU}$, (3.11)

for $t_2$ equation

$t_2 = t + \Delta_{\text{transceiver}_{rx}} + \Delta_{FDU} + \Delta_{\text{propagation}}$ (3.12)

applies.
Figure 3.17: Delay of hardware timestamping with MCAN delay measurement

Figure 3.17 shows the time sequence. The timestamp here is shifted back by the measured delay. For the timestamps $t_1$ and $t_2$ the equation

$$t_1 = t - \Delta_{\text{transceiver}_{tx}} + \Delta_{\text{FDU}}$$

(3.14)

and

$$t_2 = t - \Delta_{\text{transceiver}_{tx}} + \Delta_{\text{FDU}} + \Delta_{\text{propagation}}$$

(3.15)

follows.

For the difference $\Delta_{\text{HWTS,corrected}}$

$$\Delta_{\text{HWTS}} = t_2 - t_1 = \Delta_{\text{transceiver}_{tx},\text{master}} - \Delta_{\text{transceiver}_{tx},\text{slave}} + \Delta_{\text{FDU,slave}} - \Delta_{\text{FDU,master}} + \Delta_{\text{propagation}}$$

(3.16)

follows.
If comparing 3.13 and 3.16, the only difference between the two equations will be the transceiver term. Instead of the receive delay the transmit delay now influences the overall delay. Assuming that the transceiver delays are nearly constant, there is no added value in including the transceiver delay. Furthermore a CAN FD message must be sent for a delay measurement. For the synchronization master this is only the case if the messages are sent as CAN FD messages. However, this leads to the loss of compatibility with classic CAN. For the synchronization slave separate messages would have to be sent to get a measurement.

Since the inclusion of the delay measurement promises no advantages in the accuracy of the timestamp and creates disadvantages in terms of compatibility and effort, the timestamp without delay measurement is used.
4 Approach

This chapter gives an overview of the methods used to achieve the highest possible precision with AUTOSAR time synchronization over CAN. First, the synchronization model with the EB time synchronization IP core is discussed. This is divided into rates and offset correction. The reason for the division is that the rate should not be influenced by the offset correction. If no more synchronization messages are received, this ensures that the rate does not contain any offset correction components. Based on the model several approaches for improvement are presented, which are evaluated in the following chapter.

4.1 General model

In order to define a generic model for clock synchronization, the input parameters must first be defined. By exchanging messages with a master, a slave always receives two timestamps. One timestamp is taken by the master clock, the other by the slave clock. In the specific case this means that the master receives the timestamp \( t_{\text{Master},i} \) and the slave the timestamp \( t_{\text{Slave},i} \). The EB time synchronization IP core also contains the register \( t_{\text{Offset},i} \). From these three values the following condition

\[
 t_{\text{Master},i} = t_{\text{Slave},i} + t_{\text{Offset},i} \quad (4.1)
\]

can be formulated. This condition describes the synchronized target state and applies to the timestamp point \( i \). Figure 4.1 shows again the times at which the timestamps are generated. Master and slave time are in this context the time domains in which the timestamps are generated. The timestamps \( t_{\text{master},i} \) are thus generated by the local clock on the master device. The timestamps \( t_{\text{Slave},i} \) are generated from the time at the slave.

For a rate correction the two timestamps \( t_{\text{Master},i} \), \( t_{\text{Slave},i} \) and \( t_{\text{Offset},i} \) are not sufficient. This also needs past values to calculate time differences. These are \( t_{\text{Master},i-1} \) and \( t_{\text{Slave},i-1} \), they are the timestamps from the previous round. With these values the time difference between the points can be calculated. From these differences the condition

\[
 t_{\text{Master},i} - t_{\text{Master},i-1} = t_{\text{Slave},i} - t_{\text{Slave},i-1} \quad (4.2)
\]

can be formulated, where master and slave run with the same frequency. In this state they are syntonized.
approach

4.1.1 Offset correction

The offset and rate correction should function largely independently of each other. Therefore it is necessary that both mechanisms know about the parameters of the other one. This is done with the parameter applied difference and will be discussed in the following chapters. For the offset correction the condition 4.1 must be fulfilled as good as possible, for the rate correction the condition 4.2 applies.

Figure 4.2 shows an overview of the main components of the offset correction.

The following input parameters are required every sync round for the offset correction:

- \( t_{\text{master},i} \)
  Actual timestamp from master delivered over AUTOSAR time synchronization over CAN.

- \( t_{\text{slave},i} \)
  Actual timestamp taken at reception of SYNC message.

- \( t_{\text{offset},i-1} \)
  Actual offset value before offset correction was applied.

- Actual tick length
  Actual tick length calculated by rate correction mechanism.
Approach

Figure 4.2: Offset correction structure

- Interval
  Interval in which the rate adjustment should be applied.

Following parameters are required for the configuration of the offset correction:

- Rate adjustment threshold
  Threshold for decider.

- Maximum rate change
  Value in parts per billion, which determines how much rate change is allowed by the offset adjustment.

The mechanism generates the following output parameters:

- Temporary tick len
  Temporary tick length to be applied by the rate adjustment mechanism.

- Number of temporary ticks
  Number of ticks in which Temporary tick len should be applied.

- Applied difference
  Difference to be corrected by the rate adjustment mechanism.

The process consists of the following steps:
1. Calculate difference

From $t_{Master,i}$, $t_{Slave,i}$ and $t_{Offset,i-1}$ a deviation $\Delta$ according to

$$\Delta = t_{Slave,i} + t_{Offset,i-1} - t_{Master,i}$$  \hspace{1cm} (4.3)

is calculated.

2. Decide

The calculated difference is checked for validity and sent to a decider. The decision made is based on the size of the difference which offset correction measure should be taken. The Rate adjustment threshold determines how fast the offset is corrected. A good value for the decider makes sure that an offset jump only occurs with real time jumps of the syncmaster.

3. Apply

(a) Offset adjustment

If the value is above the Rate adjustment threshold, the offset register is changed. The change follows directly from the difference. From equation 4.1 and 4.3 then follows

$$t_{Offset,i} = t_{Offset,i-1} - \Delta.$$  \hspace{1cm} (4.4)

(b) Temporary rate adjustment

If the value lies below Rate adjustment threshold, the offset is corrected by the rate. This requires the current rate, which is determined by the rate correction mechanism. Furthermore, an interval is needed in which the offset is to be corrected. This interval must end before the next timestamp point, otherwise errors in the rate correction will occur. Furthermore, Maximum rate change is required, which determines how fast an offset can be corrected. First, Maximum ticks that can be applied are determined from the interval using

$$\text{Maximum ticks} = \frac{\text{Interval} \ll 26}{\text{Actual tick length}}.$$  \hspace{1cm} (4.5)

The shift operation converts the Interval into the format of the tick length. Next step is to calculate the Maximum difference that can be corrected in the given interval. This value depends on the Maximum rate change. It is calculated according to

$$\text{Maximum difference} = \frac{\text{interval} \times \text{Maximum rate change}}{1000000000}.$$  \hspace{1cm} (4.6)

The next step is to compare if the $\Delta$ is greater than the Maximum difference. If yes, the calculated Maximum ticks are set as number of Number of temporary ticks. If no, the Number of temporary ticks are determined with

$$\text{Number of temporary ticks} = \frac{|\Delta| \times 1000000000}{\text{Maximum rate change}}.$$  \hspace{1cm} (4.7)

Afterwards the Change per tick is determined which can be applied. This is calculated as
Approach

\[
\text{Change per tick} = \text{sign}(\Delta) \times \frac{\text{Actual tick length} \times \text{Maximum rate change}}{1000000000}. \tag{4.8}
\]

From this

\[
\text{Temporary tick length} = \text{Actual tick length} + \text{Change per tick} \tag{4.9}
\]

and

\[
\text{Applied diff} = \text{Change per tick} \times \text{Number of temporary ticks} \tag{4.10}
\]

is calculated and applied.

Figure 4.3 shows the time behavior of the offset adjustment. For the total time it looks like a time jump, but the jump is only in the offset. The normal timestamp remains monotonically increasing at a constant rate. This is visualized by the Actual rate.

![Offset adjustment diagram](image)

**Figure 4.3**: Offset adjustment

Figure 4.4 shows the time behavior of the rate adjustment. Actual rate shows the rate generated by the Actual Tick Length. One can see that the Interval is shorter than the sync period. As already mentioned, this is to not influence the rate correction. The graph shows the two possible cases, in the first case the Interval is not sufficient to correct the whole Δ. Therefore at the next timestamp point there is a remaining error above which the next sync period will be completely corrected.
4.1.2 Rate correction

Figure 4.5 shows an overview of the basic model of the rate correction. This mechanism is purely for determining and applying a stable rate. Therefore, this measure does not compensate for offsets and has only the goal of syntonizing the devices.

The following input parameter are required:

- $t_{\text{Master},i}$
- $t_{\text{Master},i-1}$
  Master timestamp from the previous synchronization round.
- $t_{\text{Slave},i}$
- $t_{\text{Slave},i-1}$
  Slave timestamp from the previous synchronization round.
- Applied difference
  Applied difference which was applied in the previous sync period. The calculated value from the offset correction is always applied in the next sync period.
- Previous tick length
  Tick length which was applied in the previous sync period.

The mechanism generates the following output parameters:
Approach

Figure 4.5: Rate correction structure

- Actual tick length
  New tick length to be applied in the next sync period. This is also the value which is used by the offset correction.

The process consists of the following steps:

1. (a) **Calculate slave period**
   A period is calculated from the arrival timestamps of the Sync messages. Since the nominal rate of the system is required here, the value which is corrected by the offset adjustment must be included. This is shown graphically in the figure 4.4 with the actual rate. The slave period is calculated by evaluating

   \[ Period_{Slave} = t_{Slave,i} - \text{Applied difference} - t_{Slave,i-1}. \]  

   \[ (4.11) \]

(b) **Calculate master period**
   From the timestamps transported in the SYNC and FUP messages the reference period is calculated according to

   \[ Period_{Master} = t_{Master,i} - t_{Master,i-1} \]

   \[ (4.12) \]

   for the target rate.

2. **Calculate ratio**
   The ratio is now determined from both periods. This ratio indicates by how much the local clock differs from the master clock with respect to the rate. This is calculated according to

   \[ \text{Ratio} = \frac{Period_{Master}}{Period_{Slave}}. \]  

   \[ (4.13) \]
3. **Calculate new tick length**

In the last step the calculated ratio is applied to the current rate and a new rate for the next sync period is calculated according to

\[
\text{Actual tick length} = \text{Ratio} \times \text{Previous tick length}. \tag{4.14}
\]

The calculated rate is checked for plausibility. This is done by a comparison with a limit value.

The described process is the basis for the rate correction in this system. It is trusted that valid input data is always available. For an ideal system this is true, but in case of variances, for example in the timestamp point, the rate can vary very much.

To prevent these problems, the following chapters present strategies to either smooth or reduce such variances. It is assumed that the rate changes continuously. This assumption can be justified by the fact that the main reasons for variations in oscillator frequency are aging and temperature influences. These processes are slow compared to the oscillator frequency. Rapid changes in the frequency would mean an error case.

In the following chapters the model is first implemented purely in software, as a next step filters for smoothing the variances are presented. In order to reduce the variances, hardware timestamping is used and then combined with the presented filters.

### 4.2 Software solution

The presented solutions are executed directly as tasks on the real time core. Since this core works with a cooperating scheduler, the exact time of execution is strongly dependent on the system load. The first approach relies entirely on software and does not use hardware support.

The master works according to the following scheme:

- First the current system time is read out via a software interface with a resolution of 25 ns. The part designating the seconds of this timestamp is then taken and sent as part of the SYNC message.
- As soon as the message is sent, the time of sending is determined by a callback. This callback is triggered as soon as the frame has been received again by the controller and read out by the receive task.
- The nanosecond part and a possible second overflow is then sent with the FUP message.

In both cases the determined times originate from the corrected nanosecond counter. The transmitted timestamp is divided into two parts. The first part, which is sent via the SYNC message, contains only the seconds. The second part contains the nanosecond part and a possible seconds overflow. The reason for the division is the limitation of the message length to eight bytes at the CAN bus. If one would transmit a whole 64Bit timestamp with a message, there would be no more space for a header and the message could therefore not be assigned to any protocol.

The slave system operates according to the following scheme:
In the slave system the messages are received via callbacks. When the callback is called a timestamp is taken first. After checking the message it is either discarded, for example in case of faulty messages or processed further. This happens in case of received SYNC messages.

After receiving a SYNC message the system waits for a FUP message with the same sequence number.

The receive timestamp is stored as $t_{\text{Slave},i}$. The determined timestamp from the messages is stored as $t_{\text{Master},i}$. Existing timestamps are stored as timestamps of the last round.

Based on these values the rates and offset correction according to 4.1 is calculated. If no past values are known yet, only an offset correction is performed.

For the value actual tick length the nominal tick length is used in the first round.

The calculated values are provided directly to the FPGA and applied immediately. The calculated value for the applied difference is stored for the next round and used by the rate correction.

This solution directly implements the AUTOSAR time synchronization over CAN protocol. The advantage of this implementation is its simplicity. No additional hardware resources are used except for the EB clocks synchronization ip core.

The disadvantage, however, is the high expected inaccuracy. This is because the determination of timestamps is subject to great fluctuations. This is partly due to the cooperative scheduler and partly to the delay caused by the time query. These variances lead to an increased offset and an inaccurate rate correction, because the time intervals between the timestamp points cannot be measured exactly.

Exactly this problem will be treated with the next approach. By filtering, variances in the timestamp point shall be reduced. These measures should lead to an improved rate correction.

### 4.3 Software solution with filtering

With this solution the calculated new rate is filtered. This should smooth the calculated data and suppress possible outliers. Due to the variance of the timestamps and possible longer software processing times, data is generated that does not reflect the true time relationships. However, only the distances between the timestamps are relevant for the rate correction. The idea of the filtering is that these distances give the correct value on average. Therefore, different mean value filters are analyzed here. The assumption is based on the analysis of section 3.6.

A moving average, an exponentially weighted average and a moving median filter are examined. In addition, a Kalman filter is considered. The possible solutions are checked on the basis of real measured data. For this purpose, timestamps were recorded simultaneously by two synchronized devices. From these timestamps the intervals were calculated and put into relation. The figures are taken from this data. The value 1 corresponds to the case that the two devices run completely synchronously. In this case, the intervals of both devices have the same duration. The peaks in the raw data represent intervals with deviating lengths. These are primarily caused by quantization effects of the timestamp, which result from the granularity of the scheduler and the clock.

The new structure of the rate correction is shown in figure 4.6. Instead of the directly calculated rate, the filtered rate is now applied to the system.
4.3.1 Averaging filters

The easiest way to filter the noise components from a signal is to calculate the moving average of a signal [Smi03]. This takes $N$ data points and calculates the mean value. It is very well suited to filter high frequency noise from the signal. The filter width $N$ is the determining factor for the rate of change of the output. A new filter value can be calculated using the equation

$$x_{\text{filtered},i} = \frac{1}{N} \sum_{n=0}^{N} x_{i-N}. \hspace{1cm} (4.15)$$

Where $x_{\text{filtered},i}$ corresponds to the output and $x_{i-N}$ to the data points.

In general it can be said that a larger filter width leads to slow changes in the filter value. The advantage is that large outliers are averaged away. The disadvantage of large filter widths is that they consume a lot of memory and resources. To get an idea of the influence of different filter widths, a simulation is used.

The data basis for the simulation is provided by timestamps generated by two different devices at the same time. The times between the timestamps are set in relation to each other and serve as filter input data.

Figure 4.7 shows the behavior of the filter with different widths. From the diagram it can be concluded that wider filters are necessary for rate correction, because they filter outliers better and the filtered value itself is not influenced too much. However, large filter widths have the disadvantage that they consume a lot of memory.

4.3.2 Exponentially weighted average

One way to save memory and still keep the weighting on the current values is the exponentially weighted average filter [AID16]. This filter always weights the current value more than the history.
The weighting is set by the filter width. To get a valid basis for the filtering, the filter is initialized with a mean value.

For a filter width $M$, the equation
\[ x_{\text{filtered},i} = x_i + x_{\text{filtered},i-1} \left( \frac{i}{i-1} - 1 \right) \]
(4.16)

applies to the first $M$ values.

After that the exponentially weighted part is used. The weighting factor is calculated according to
\[ c_{\text{filter}} = e^{-\frac{1}{M}}. \]
(4.17)

The new filter values result from
\[ x_{\text{filtered},i} = x_i (c_{\text{filter}} - 1) + x_{\text{filtered},i-1} \cdot c_{\text{filter}}. \]
(4.18)

The same data basis was used as for the moving average filter.

Figure 4.8 shows the exponentially weighted average filter applied to noisy data. Here small filter widths are an advantage, because the value can be followed faster and the past mean value is not weighted so strongly. One can see that if the filter width is too large, the general average will prevail. This has the potential disadvantage that slow rate changes are not detected. The advantage of this filter is its memory efficiency, because only the last value has to be stored. With this filter similar results as with the moving average filter can be achieved.

### 4.3.3 Moving median

A variant to exclude outliers completely is a moving median filter [TJ19]. It works especially well if the data consists of valid measurement points and outliers. In this case the outliers are removed. For noisy data the situation is not so favorable anymore, because there is no averaging.
The filter requires a data set with the entire filter width. The following steps are necessary for filtering.

- Remove the oldest value
- Adding the new value
- Sorting the data
- Selection of the median

The implementation of such a filter requires high hardware resources. Not only the whole data set has to be stored, but also a reference to the oldest value. In addition, sorting the data requires a lot of CPU power unlike the other filters.

Figure 4.9 shows the moving median filter applied to the noisy data. It is clearly visible that there is no averaging. This can be seen from the abrupt changes of the values.

4.3.4 Kalman filter

Another way to deal with noisy data is the Kalman filter [Kal60]. This chooses a completely different approach. States are estimated on the basis of a model and measurements. The statistical properties of the measurement and the model are taken into account. The Kalman filter is based on a minimum variance estimator. This section deals only with the rough analysis, the usability of the Kalman filter for this application. A detailed modeling and analysis would be a research works on its own.
To design a Kalman filter, one needs an underlying model. For this case, a simple model of a clock increasing at a constant rate is used. The model consists of an iteration equation for the current time

\[ t_{i+1} = t_i + R_i \times dt. \]  

(4.19)

And one for the rate

\[ R_{i+1} = R_i. \]  

(4.20)

Where \( R_i \) is the clock rate and \( dt \) is the time since last timestamp. The goal is to estimate the rate based on new timestamps. It also uses known covariances of the measurement and model error. Based on this data a new state and a new error covariance is estimated.

Already the model for this application shows some difficulties. In the used system no floating point variables are available, this would require either a scaling of the model. This would either require a scaling of the model, but would cause problems with regard to the value range, which is limited by this. So a trade off between range of values and accuracy of the calculation would have to be found. A further problem arises with the determination of the parameters. These would have to represent the clock and the errors as well as possible. For this one would have to extend the model either by an error term or to adjust the covariances of the model error very well to the model.

Since a solution of these problems would divert the focus of this work too far from the actual problem, a more detailed evaluation is renounced.

### 4.3.5 Comparison

The choice of the right filter depends on the data to be expected. For the approach with pure software timestamps, large fluctuations in the timestamps are to be expected. The real rate...
changes very slowly if at all. Therefore one has to choose a filter that reacts well to slow changes and filters fast changes without distorting the value too much. Another criterion is the resource requirements of the filter. Because we are working on an embedded system with limited memory, the demand must not be too high to not disturb other applications.

The moving median filter is omitted by both criteria. It requires valid data as a basis. This is not given by the large uncertainty in the timestamp point. It is also the filter with the highest demand on resources. It needs the most memory as well as the most CPU resources.

The moving average filter gives similar results as the exponentially weighted average filter. The latter has the big advantage that it is much easier and more efficient to implement. It only needs an additional value as history. The filter coefficient can be pre-calculated, which saves even more resources.

The exponentially weighted average filter is the best choice, because it fulfills the criteria best.

A suitable coefficient must now be found for the selected filter. A compromise between averaging and response time must be found. A choice based on the test data would lead to a very high filter width. The filter would then turn into an ordinary averaging filter. But since the rate can change over time due to temperature effects and aging, this does not provide a good choice. The choice of filter width must also take into account that a higher synchronization period already results in averaging.

However, a too small filter width would lead to too strong reaction to timestamp errors. This would result in a wrong value for the rate. This in turn leads to an increase in the error, since the rate is only ever adjusted at synchronization times.

Figure 4.10 shows the relationship between filter width and coefficient. The relationship was determined from equation 4.17. Due to the exponential relationship between filter width and filter coefficient, the change of the coefficient decreases strongly with increasing width. Therefore, it makes little sense to choose a very high width, since this only prolongs the initialization time of the filter.

For these reasons a filter width of 15 was chosen. This is already in the range of the exponential curve where the coefficient changes only little.

The use of filtering for rate correction tries to reduce the influence of the variances in the timestamp taking. The advantage is that the solution still works without hardware support. However, the achievable accuracy is limited by the use of software timestamps. Therefore the next step is to improve the accuracy of software based timestamps.

4.4 Improving software timestamping

A high jitter is expected from the timestamps of pure software solutions due to the scheduling delay. This delay is directly related to the polling period of the CAN receive task. But since the scheduler works cooperatively, the exact delay cannot be determined.

One measure to reduce jitter is to bring the timestamp closer to the hardware. This means that the time request for received CAN messages is no longer done in the application itself, but already in the lower levels of the operating system, more precisely in the hardware driver.
Figure 4.10: Relation of filter width and coefficient

Other solutions with software timestamping use interrupts for this purpose. An interrupt is triggered as soon as a message is received from the CAN controller. This is not possible on the EBX200 because the messages are received by the CAN controller via a polling routine. In addition, an interrupt would limit the real-time capability of the system, since it does not fit to the fixed time schedule.

Therefore the approach is to improve the polling routine. This can be achieved by several measures, including the following:

- **Reduction of the task period**

  By reducing the task period CAN messages are received more often by the CAN controller. This reduces the time interval between receiving the message in the controller and calling the software callback. Under ideal conditions this measure promises a linear improvement of the accuracy of the timestamp. In this case, ideal conditions mean that the task is always executed at the scheduled time. This measure has a natural limit and this is the execution time of the receive task. The default value of the period is 500 microseconds, the minimum value is 100 microseconds.

- **Increasing the task priority**
Approach

The different tasks in the scheduler have different priorities. They decide which task will be executed next, if there are several potential candidates. This measure promises a reduction of jitter, especially if the scheduler has many tasks in its queue. However, it does not protect against delays caused by tasks already running.

- Removing other Tasks
  Removing other tasks ensures that no other tasks delay the execution of the CAN receive task. This promises an improvement, especially in connection with the two measures already mentioned.

- Timestamping in the driver
  Another possibility is to timestamp the CAN driver already. In the case of the EBX200 Platform the CAN driver consists of several access functions for the CAN controller. These functions receive messages from the memory of the CAN controller and then trigger the corresponding callbacks. Instead of waiting for the callbacks, the timestamp can be generated directly in the access functions. This leads to a reduction of the software dependency and thus to a reduction of the variable delay.

The measures mentioned make only limited sense in themselves. Only in combination of these measures can a substantial improvement be expected. The reason for this is that the measures influence each other.

The most promising measure here is the reduction of the task period. An improvement can only be achieved if the receiving task is not disturbed by any other task. Increasing the task priority helps here. This means that the receiving task is always given priority in the case of a decision, which increases the chance that the task will be executed on time. Removing other tasks also has a great influence on the first measure. This reduces potential delays caused by other tasks. The relocation of the timestamp in the driver is a measure in itself, since it is not influenced by the scheduler.

These measures are used to improve software timestamping and bring the resulting solution closer to the precision of other software-based solutions. However, it is not expected that these measures will provide a solution with the same accuracy as interrupt based solutions.

4.5 Hardware solution

For the hardware solution the software timestamps are replaced by hardware timestamps. Thus the process loses its dependence on the software processing times. The reason for this is that the timestamps for sending and receiving messages are recorded directly in the FPGA. The structure of the rates and offset correction does not change. The confidence in the recorded and transmitted timestamps increases enormously.

The procedure for processing the messages is different from the software solution. The reason for this is that the software is no longer responsible for the generation of timestamps. Thus, the whole task is no longer time-critical and is limited to the processing of information.

The master carries out the following steps:
Approach

- The current time is determined via software interface. The seconds part of it is calculated and sent with the SYNC message.

- Frames are recorded on the CAN bus during the entire process. Sent messages are treated like received messages as soon as they have been sent successfully. So it can be determined by receiving the sent message whether a message was sent successfully. Not only the pure CAN frame is received, but also the corresponding EBHSCR header (See section 3.4). This header contains two timestamps, the start timestamp which is generated after the start of frame bit and the stop timestamp which is generated after the end of frame bit. If now the sent SYNC message is received, the stop timestamp is stored from it.

- The nanosecond part of the stop timestamp is written to the FUP message with possible second overflow.

The slave works mostly with the same scheme as in the software solution. The only difference is the recording of the timestamp. While the slave process is active, CAN frames are measured continuously as encapsulated EBHSCR packets. The received packets are parsed and checked if they are SYNC or FUP messages. If the packets are valid the stop timestamp of the SYNC message is used instead of the software timestamp of the callback. This makes the slave independent from the time the packet was received.

This solution implements, like the software solution, the message format of AUTOSAR time synchronization over CAN. It has the big advantage that the timestamps are always received with almost the same small delay because the process is completely implemented in the FPGA. The downside is that the special hardware is needed for the recording. In this case the same hardware is used for capture replay tasks, so existing hardware resources are used.

The main source of the remaining variance is the resolution of the timer. The timer is incremented at a nominal rate of 25 nanoseconds per tick. This results in inaccuracies in the timestamp recording. This problem should be counteracted with suitable filtering.

4.6 Hardware solution with filtering

Hardware timestamping is expected to significantly reduce jitter and delay. Additional filtering should further stabilize the rate. This assumption follows from the assumption that the granularity of the local clock causes a quantization error. The influence of the error should be reduced by suitable statistical filtering.

The procedure for processing the messages is similar to the hardware solution. However, like the software solution with filtering, a filter is inserted before the rate correction is applied. The structure corresponds to the one shown in figure 4.6.

An exponentially weighted average filter is also used here. The reasons are the same as for the software solution with filtering.

This solution is generally expected to be more stable than with pure hardware timestamping without filtering. For the precision this means that the distribution of individual measuring points will probably have a smaller width. Furthermore, it is expected that the system will operate longer without resynchronization.
Approach

An error factor that is not eliminated by this solution is the propagation delay. Due to this error, the mean value of precision can never be zero, because the timestamp is always delayed by this delay. A solution to counteract this error is to perform a delay measurement. The AUTOSAR time synchronization over CAN standard does not provide a solution for this. Therefore the next solution is outside the standard.

4.7 Delay measurement

The biggest obvious weakness of the AUTOSAR standard is that the propagation delay in synchronization is not corrected. This means that there is always a residual error in precision, since the convergence interval can never be smaller than the delay. To measure and correct the influence of the delay, the solution is extended by a delay measurement.

Two additional messages are introduced for this measurement. A delay request message and a delay response message. This type of delay measurement is based on the end-to-end measurement of PTP[IEE20]. The delay request message is sent by the slave and is timestamped when the slave sends it. As soon as the master receives the message, the message is also timestamped there. Thus we get the send timestamp t4 and the receive timestamp t5. The timestamp t5 is then sent to the slave with the delay response message. Figure 4.11 illustrates the message exchange with delay measurement.

For the delay response message a standard CAN message with 8 bytes length was selected. The content is irrelevant for this message, because it is only about the timestamps. For the delay response message a CANFD message with 16 bytes length was chosen, the first eight bytes are the identifier for the message and the last eight bytes contain the timestamp t5. To transmit this timestamp with a standard CAN message, an additional delay response follow up message would be necessary, because the available eight bytes are not sufficient for timestamp and identifier. This is not used here, because all involved controllers support CANFD and it is only about the delay measurement.

The delay measurement uses the fact that two timestamps are already known by the transmission of the Sync message. The remaining timestamps are transmitted by the new messages. After complete transmission the timestamps t2, t3, t4, and t5 are known. Assuming that the propagation delay is the same in both directions, the end-to-end delay is calculated according to

\[ \Delta_{End-to-End} = \frac{(t5 - t2) - (t4 - t3)}{2}. \] (4.21)

The calculated delay is filtered with an exponentially weighted average filter to reduce the influence of the clock granularity. This filter should result in a constant value for the delay. This filtered delay is then used to correct the receive timestamp t3 by exactly this delay. The corrected timestamp \( t_{3_{corr}} \) is then calculated according to

\[ t_{3_{corr}} = t3 - \Delta_{End-to-End}. \] (4.22)

Instead of the receive timestamp t3 the corrected timestamp \( t_{3_{corr}} \) is now used for synchronization.

It is expected that the measured values for the deviation between master and slave will have an average value of zero. The average value without delay measurement should therefore correspond
Figure 4.11: CAN synchronization with delay measurement

to the measured end-to-end delay. This measure reduces the convergence interval, which is expected to improve precision.
5 Evaluation

In this chapter the presented solutions shall be evaluated. The main goal is to determine the precision with which two devices can be synchronized. This should happen for all solutions with different methods and scenarios.

First, the different methods are presented. Subsequently, the measurement results of the individual approaches are presented and discussed.

5.1 Measurement methods

In this work, two independent methods are used to determine the precision. In the first method, the clock signal is measured directly with an oscilloscope. In the second variant, timestamps are generated by voltage signals.

For all measurements the CAN bus is operated with a bit rate of 500kBits.

5.1.1 Measurement via timer GPIO

The function of timer GPIOs is described in more detail in section 3.5.2.

Figure 5.1 shows the measurement setup. Two EBX200 devices are synchronized with each other via CAN. One output pin of every device of the timer GPIO is connected to the oscilloscope. The phase position between the two signals is measured. This can be directly evaluated as offset of the two clocks.

The measurement is based on the fact that the internal timers of the EBX200 devices should run synchronously. Therefore it can be expected that the edges of the two timers occur at exactly the same time for synchronized devices. The maximum deviation of the edge timing is interpreted as the precision of the system. The measurements are performed on bit 31 of the corrected nanosecond timer. For this type of measurement, the offset register is not set, because this would cause the edges to be offset by the value of the offset register. This would make the measurement more difficult.

A potential problem with this method is that it takes a certain amount of time for the signal to reach the output. However, this delay is expected to be the same and constant for both devices. This assumption is based on the fact that the complete signal path is implemented in hardware.
The disadvantage of this method is that the distribution of the data cannot be inferred from the measurement. The reason for this is that the oscilloscope does not allow access to individual measured values of the jitter. The data are statistically evaluated and a minimum and mean value is available. In addition, a standard deviation is calculated, but this value should be treated with caution, since it is always assumed that a normal distribution is present. This kind of distribution is expected from the measurement data, but without exact measurement this is only an assumption. For this reason, an additional measurement is made with DETI.

5.1.2 Measurement via DETI

The function of DETI is described in more detail in 3.5.1.

Figure 5.2 shows the measurement setup. Two EBX200s are synchronized with each other via CAN. The two devices are also connected to a measuring computer. This computer records the data generated by DETI. The trigger source is a function generator which generates a periodic square wave signal. The output of the function generator is connected to a DETI port on both EBX200 devices. A timestamp is now generated for each rising edge, which is sent to the host PC.

On the host PC, a timestamp from the master and slave device is now available for each edge. The offset is then calculated from these timestamps

\[
\text{Offset}_i = t_{\text{Slave},i} - t_{\text{Master},i}.
\]

This approach samples the local clocks periodically. Thus, the deviation of the clocks can be determined at arbitrary points in time.

The advantage of this measurement method is that it allows a statistical evaluation. This is possible because all timestamps of the events are available as data. Thus, the actual distribution...
of the deviation between master and slave can be derived by this method, and the assumption of the normal distribution can be checked.

A potential weakness of this method is, as with the method with timer GPIOs, the delay. In this case it is the delay between the arrival of signal from the function generator until the generation of the timestamp. Here again it is assumed that the delays are constant, because this feature is completely implemented in hardware. In addition, it is ensured that the cables between the function generator and the devices have the same length.

The frequency used in this measurement only plays a minor role. It only determines the frequency of the measurement packages. However, it is not suitable as an absolute time reference, since the internal crystal oscillator of the EBX200 has a higher stability than that of the function generator.

The period of the pulses was chosen to be 10 Hz and 2000 data points were always recorded. The measurement period is thus approximately 200 seconds. The internal oscillators have a deviation of 50ppm. In case one quartz deviates by 50ppm and the other by -50ppm, the deviation is 20 milliseconds. This value can be seen as a reference value for the deviation. It is expected that the measured deviation is well below this value.

In general, it is expected that both measurement methods provide almost the same results. If, contrary to expectations, this does not occur, the methods must be reevaluated.
5.2 Measurement results

5.2.1 Software solution

In this section, the measurement results of the software solution are analyzed and processed. The CAN bus was only used for synchronization between master and slave. Therefore there is no additional load on the bus.

Figure 5.3 shows a screen shot of the timer GPIO measurement. The green line represents the observed pin of the master, the yellow line represents the observed pin of the slave Figure 5.4 represents a histogram of all measured data using DETI.

![Timer GPIO measurement of software solution](image)

Figure 5.3: Timer GPIO measurement of software solution

Table 5.1 summarizes the results of the two measurements. Statistical characteristic values were also determined from the data of the DETI measurement. It is noticeable that the values of the two measurements diverge. This is due to the fact that the two measurements were performed independently of each other. The reason for the large scatter of the values is the scheduler, especially the large period of 500 µs. The reason why this has such a strong effect is because the schedulers are not synchronized with each other. An explanation with examples can be found in chapter 3.6.

The distribution from Figure 5.4 shows several maxima. One reason for this is the rate and offset correction. Each synchronization round a new rate is calculated and based on this rate a temporary rate is then calculated to compensate for the offset. This temporary rate is never
applied over the entire sync interval and tends to be larger than the base rate. The hills in the distribution can thus be partially explained by the base rate, since it is slower, the offset does not change as quickly due to it. The valleys are thus the areas where the more temporary rate acts. But this is not the only reason for this distribution. The scheduler contributes by its period to an additional quantization of the timestamp, with which the input data can change fast from one extreme to the other. These effects have a particularly strong impact on the rate and offset correction.

The values for the precision of the pure software solution are very large with about 415µs. The large value confirms the assumption that the cooperative scheduler has a high influence on the precision of the timestamps. Due to these inaccuracies, an erroneous value is calculated by the rate correction. This circumstance contributes a large part to the measurement result. It is expected that the filter will at least eliminate the influence of the erroneous rate.
5.2.2 Software solution with filtering

In this section, the measurement results of the software solution approach with rate filter are discussed. The CAN bus was again only used for synchronization, therefore with no additional bus load. An exponentially weighted average filter was used as the rate filter.

Figure 5.5 shows the measurement by timer GPIO. The green line represents the master and the yellow line the slave. Figure 5.6 shows the histogram of the DETI data. Table 5.2 summarizes the results of the two measurements.

![Timer GPIO measurement of software solution with filtering](image)

**Figure 5.5:** Timer GPIO measurement of software solution with filtering

<table>
<thead>
<tr>
<th>Method</th>
<th>Synchronization period</th>
<th>Measured Points</th>
<th>Max(Offset)</th>
<th>Min(Offset)</th>
<th>Mean(Offset)</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer GPIO</td>
<td>3s</td>
<td>5 766 000</td>
<td>-180µs</td>
<td>-224µs</td>
<td>-209µs</td>
<td>10µs</td>
</tr>
<tr>
<td>DETI</td>
<td>3s</td>
<td>2 000</td>
<td>-201µs</td>
<td>-242µs</td>
<td>-226µs</td>
<td>11µs</td>
</tr>
</tbody>
</table>

**Table 5.2:** Measurement results of software solution with filtering

The distribution in figure 5.5 again shows hills and valleys. As with the pure software solution, this can be attributed to the rates and offset correction. But since the rate is not subject to such large changes due to the filter, the peaks tend to be narrower. It is also noticeable here that the distribution does not include the zero point and generally has a different mean value than the straightforward software solution without filtering. This can be attributed to the more
stable rate and thus the more stable phasing of the schedulers with respect to each other. The two schedulers run independently, but depend on the local clock. Because if the schedulers have a constant phasing, the error between master and slave timestamp is also constant. This fact is explained in chapter 3.6, specifically by equation 3.6, with which assumptions of nearly constant scheduling delays are explained. Another reason for the distribution is the quantization effect of the scheduler.

The value of 243µs for precision is still very large. The assumption that the filter leads to a more stable rate could be confirmed. This manifests itself by the substantially smaller width of the distribution, as with the variant without filter. However, there is a non-vanishing offset of more than 200µs in this solution. This can be justified by the different delays which are generated by the scheduler among other things. It is expected that the influence of the schedulers, and thus the non-vanishing offset, can be mitigated by improving the software timing.

5.2.3 Improved software timestamping

In this section, the measurement results of the improved software solution approach with and without rate filter are discussed. The CAN bus was again only used for synchronization.

The period of the scheduler could be reduced from 500µs to 100µs. However, this represents the lower limit for the system used. This approach is only measured with DETI. The reason for that is the unavailability of the oscilloscope due to COVID19. Figure 5.7 shows the results of the measurement of the improved software solution. Figure 5.8 shows the measurement results of the improved software solution with rate filtering.
The distribution from figure 5.7 again shows some peaks. These can be attributed to periods where the rate of the master is very similar to the rate of the slave, which means that the offset does not change very much in such periods. Since the rate changes here again more strongly in comparison to filtered solutions, the phase position of the schedulers is not constant to each other and thus the offset is distributed over a larger range.

The distribution from Figure 5.8 again shows hills and valleys and has no mean value around zero. The reasons for this are the same as for the software solution with filter.

![Figure 5.7: DETI measurement of improved software solution](image)

Table 5.3 summarizes the results. This shows the large influence of the scheduler on the rate correction. By optimizing the scheduler, the offset could be reduced. It can be seen that the filter achieves the desired effect here and significantly reduces the deviation. However, a non-vanishing offset remains which can be traced back to the scheduler.

<table>
<thead>
<tr>
<th>Method</th>
<th>Sync. period</th>
<th>Measured Points</th>
<th>Max(Offset)</th>
<th>Min(Offset)</th>
<th>Mean(Offset)</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DETI</td>
<td>3s</td>
<td>2 000</td>
<td>74μs</td>
<td>-217μs</td>
<td>-69μs</td>
<td>70μs</td>
</tr>
<tr>
<td>DETI (with filtering)</td>
<td>3s</td>
<td>2 000</td>
<td>-50μs</td>
<td>-80μs</td>
<td>-63μs</td>
<td>8μs</td>
</tr>
</tbody>
</table>

Table 5.3: Measurement results of improved software solution

The original deviation of 415000 nanoseconds could be reduced by 80.74 %. A similar good precision as achieved by other state of the art works could not be achieved by a pure software solution. This is due to the large influence of the scheduler. Reducing the task period to the minimum of 100μs was not enough in this case. The problems caused by the scheduler have been
circumvented in other state of the art work by using interrupt-based message reception. This makes it difficult to compare the solutions.

As a next step, the results of the solution with hardware timestamping are discussed. This is expected to significantly improve the precision.

### 5.2.4 Hardware solution

In this section the solution with hardware timestamping is examined. The CAN bus was again only used for synchronization.

Figure 5.9 shows the results of the measurement using timer GPIO. The green line shows the master, the yellow line the slave. Figure 5.10 shows the results of the DETI measurement.

Table 5.4 summarizes the results of the two measurements. The shape of the histogram indicates that the deviations are normally distributed. The reason that the mean of the distribution is not zero is due to the propagation delay. Both measurements offer well comparable values with similar results.

As expected, the application of hardware timestamps leads to significant improvements in precision. Here, a value of less than 100 nanoseconds was already achieved without the use of a rate filter or delay compensation. This illustrates the high influence of software in the generation of timestamps. By applying the rate filter in this case, the standard deviation of the distribution is expected to decrease.
5.2.5 Hardware solution with filtering

This section discusses the results of the hardware solution with filtering. The CAN bus was only used for synchronization, accordingly no other messages were sent or received on the bus.

Figure 5.11 shows the results of the timer GPIO measurement. The green line represents the master and the yellow line the slave. Figure 5.12 shows the histogram of the DETI measurement. Table 5.5 summarizes the results of the two measurements. The two methods again determine very similar values.

The distribution in Figure 5.12 is approximately normally distributed. The reason that the mean of the distribution is not zero is due to the propagation delay.

It can be seen that a minimal reduction of the standard deviation can be achieved by using rate filtering. The expected improvement compared to the solution without filtering is so small because the measured deviations are already mostly in the order of magnitude of the granularity.
5.2.5 Hardware solution with filtering and delay compensation

In this section, we discuss the results of the hardware solution with filtering and delay compensations. However, the bus load increases minimally here, since the delay messages are also sent. In addition to the measurements without data traffic, another measurement with a loaded bus was performed to determine the influence on the synchronization precision.

Figure 5.14 shows the results of the measurement using timer GPIO without additional bus load. The green line represents the master, the yellow line the slave. Figure 5.13 shows the histogram of the data from the DETI measurement without additional bus load. Table 5.6 summarizes the

<table>
<thead>
<tr>
<th>Method</th>
<th>Synchronization period</th>
<th>Measured Points</th>
<th>Max(Offset)</th>
<th>Min(Offset)</th>
<th>Mean(Offset)</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>3s</td>
<td>4 937 000</td>
<td>17.0ns</td>
<td>-61.0ns</td>
<td>-24.0ns</td>
<td>12.4ns</td>
</tr>
<tr>
<td>DETI</td>
<td>3s</td>
<td>2 000</td>
<td>9.0ns</td>
<td>-67.0ns</td>
<td>-30.0ns</td>
<td>13.6ns</td>
</tr>
</tbody>
</table>

Table 5.5: Measurement results of hardware solution with filtering

of the clock. However, the histogram is still shifted from the zero point, which means that on average there is a permanent deviation between master and slave. This can be attributed to the transmission delay. Therefore, it is expected that the delay compensation applied in the next step will shift the mean towards 0.

5.2.6 Hardware solution with filtering and delay compensation

In this section, we discuss the results of the hardware solution with filtering and delay compensations. However, the bus load increases minimally here, since the delay messages are also sent. In addition to the measurements without data traffic, another measurement with a loaded bus was performed to determine the influence on the synchronization precision.

Figure 5.14 shows the results of the measurement using timer GPIO without additional bus load. The green line represents the master, the yellow line the slave. Figure 5.13 shows the histogram of the data from the DETI measurement without additional bus load. Table 5.6 summarizes the
results of the two measurements. Again, as expected, the two methods lead to very similar results here.

<table>
<thead>
<tr>
<th>Method</th>
<th>Synchronization period</th>
<th>Measured Points</th>
<th>Max(Offset)</th>
<th>Min(Offset)</th>
<th>Mean(Offset)</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer GPIO</td>
<td>3s</td>
<td>4 809 000</td>
<td>31.0ns</td>
<td>-44.5ns</td>
<td>-5.2ns</td>
<td>13.2ns</td>
</tr>
<tr>
<td>DETI</td>
<td>3s</td>
<td>2 000</td>
<td>31.0ns</td>
<td>-46.0ns</td>
<td>-4.3ns</td>
<td>12.8ns</td>
</tr>
</tbody>
</table>

Table 5.6: Measurement results of hardware solution with filtering and delay compensation

The application of the delay compensation has shown that the mean value moves further towards the zero point by this measure. Thus, the precision could be further improved. The values of the standard deviation are in the same range as for the approach without delay compensation, which shows that this has no influence on a stable rate correction. To test the robustness of the approach, a bus load is connected in the last step.

Adding bus load

To determine the influence of other data transmissions on the CAN bus on the synchronization, CAN frames were transmitted and received from the master to the slave during the measurement.
Evaluation

Figure 5.12: DETI measurement of hardware solution with filtering

During the entire measurement 133151 CAN frames were transmitted. The generated frames transmit 8 bytes of user data. This results in a total length of 126 bits for such a frame. The interframe space is already included. The total measurement time per measurement is 265 seconds. At a bit rate of 500kBits, this results in a bus utilization of about 13 %. This can be regarded as a minimum value, since bit stuffing is not yet taken into account here.

Figure 5.15 shows the histogram of the measurement. The results are summarized in table 5.7.

The results show that there is no major change in precision due to the increased bus load. Only the standard deviation is slightly increased. It is difficult to say in such small ranges whether this is due to the bus load, since the values are already below the granularity. Only a minimum value could be given for the bus load, since the extent of the bit stuffing is not known. Despite this limitation, the generated bus load is very low, mainly because the CAN frames were transferred from a host PC to the master and because it is an experimental setup. An alternative for this would have been to create a rest bus simulation that generates and sends the data directly on the device. However, this was prevented by the fact that it was not a setup fully integrated into the development system. This would have required full compatibility with the development tools.

Table 5.8 summarizes the results of all measurements. Once again, it can be seen that the greatest improvement was achieved by using hardware timestamping. In principle, this would already be enough to greatly improve existing solutions. It has also been shown that an improvement is possible with pure software timestamping, but this was severely limited by the scheduler. It could also be shown through the use of rate filters how the stability of synchronization solutions can be improved. This is particularly evident in the measured values with software timestamping.
**Figure 5.13:** Timer GPIO measurement of hardware solution with filtering and delay compensation

<table>
<thead>
<tr>
<th>Measurement method</th>
<th>DETI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronization period</td>
<td>3s</td>
</tr>
<tr>
<td>Bus load</td>
<td>&gt;13%</td>
</tr>
<tr>
<td>Max(offset)</td>
<td>33.0ns</td>
</tr>
<tr>
<td>Min(offset)</td>
<td>-48.0ns</td>
</tr>
<tr>
<td>Mean(offset)</td>
<td>-3.6ns</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>14.0ns</td>
</tr>
</tbody>
</table>

**Table 5.7:** Measurement with bus load
**Figure 5.14:** DETI measurement of hardware solution with filtering and delay compensation

**Figure 5.15:** DETI measurement of hardware solution with filtering and delay compensation and bus load
<table>
<thead>
<tr>
<th>Timestamping method</th>
<th>Rate filtering</th>
<th>Delay compensation</th>
<th>max(offset)</th>
<th>min(offset)</th>
<th>mean(offset)</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>No</td>
<td>No</td>
<td>224µs</td>
<td>-414µs</td>
<td>-14µs</td>
<td>164µs</td>
</tr>
<tr>
<td>Software</td>
<td>Yes</td>
<td>No</td>
<td>-180µs</td>
<td>-242µs</td>
<td>-226µs</td>
<td>11µs</td>
</tr>
<tr>
<td>Improved software</td>
<td>No</td>
<td>No</td>
<td>74µs</td>
<td>-217µs</td>
<td>-69µs</td>
<td>70µs</td>
</tr>
<tr>
<td>Improved software</td>
<td>Yes</td>
<td>No</td>
<td>-50µs</td>
<td>-80µs</td>
<td>-63µs</td>
<td>8µs</td>
</tr>
<tr>
<td>Hardware</td>
<td>No</td>
<td>No</td>
<td>21.0ns</td>
<td>-79.4ns</td>
<td>-23.5ns</td>
<td>16.9ns</td>
</tr>
<tr>
<td>Hardware</td>
<td>Yes</td>
<td>No</td>
<td>17.0ns</td>
<td>-67.0ns</td>
<td>-30.0ns</td>
<td>13.6ns</td>
</tr>
<tr>
<td>Hardware</td>
<td>Yes</td>
<td>Yes</td>
<td>31.0ns</td>
<td>-46.0ns</td>
<td>-4.3ns</td>
<td>12.8ns</td>
</tr>
</tbody>
</table>

Table 5.8: Comparison of measurement results
6 Conclusion and outlook

6.1 Summary

A strategy for precise clock synchronization via the CAN bus was developed, implemented and evaluated. The platform for this was the EBX200 hardware platform from Elektrobit. The requirements for the work were that the strategy can be used independently of the system. Therefore the solution does not require a special CAN controller. The message format is based on the AUTOSAR time synchronization over CAN standard, which promotes compatibility with other systems in the automotive environment. This standard is compatible to the standard CAN protocol and does not require any extensions like CAN FD. The main motivation was to extend the state of the art of clock synchronization over CAN in terms of precision and to bring it closer to PTP. Furthermore, it was a goal to find a strategy for clock correction that guarantees a monotonically increasing clock.

The biggest challenge for clock synchronization is accurate timestamping. Especially for software timestamping, large jitters occur here, which make the determination of exact times immensely difficult. These inaccuracies were analyzed for various scenarios. Here it was shown that the system scheduler has a particularly high influence. Delays were also analyzed for hardware solutions. Here it was shown that the propagation delay has a major influence. This is not influenced by AUTOSAR time synchronization over CAN.

Various measures were developed for the delays. One feature of the CAN FD standard is that the delay of the connected transceiver is measured when a frame is sent. One idea was to use this value for improving timestamp accuracy. However, a more detailed analysis showed that no improvement of the timestamp can be achieved by adding these values. In order to improve the software timestamps as well, a number of interventions were made in the system and the scheduler.

To reduce the influence of errors in the timestamp on the rate adaptation, several filters were evaluated and implemented. The synchronization model developed for this purpose separates the rates and offset correction into separate components. Deviations between the clocks are compensated by rate correction. This guarantees that the clock always increases monotonously. Larger deviations are compensated by a separate offset. This can be set independently of the clock.

Hardware timestamps were created in the implementation by a separate component, independent of the controller. The reason for this is on the one hand that the choice of the CAN controller is
not limited by this. But more important is the timing of the timestamp. While in many cases it is common to generate a timestamp at the beginning of a frame, AUTOSAR time synchronization over CAN requires a timestamp at the end of the frame, the reason for this being compatibility with software timestamps. These can only be generated once a message has been completely sent. Otherwise the send attempt could be prevented by the bus arbitration.

A great strength of PTP is the consideration of the propagation delay, by a delay measurement. This is not taken into account by the protocol used here. However, since this is essential for high-precision synchronization, an extension of the protocol with a delay measurement was implemented.

The evaluation of the work was done by two independent measurement methods. Firstly, the precision was determined using an oscilloscope. Signals triggered directly by the internal clock were measured. For the second method, timestamped packets were generated using digital signals. For each measure, measurements were made to determine the individual influences.

The results showed that the greatest improvement can be achieved by hardware timestamping. However, optimizations in the software also lead to improvements in precision. However, the pure software solutions are not in the range of common state of the art solutions. By using filtering, hardware timestamping and delay measurement together, a precision of 46 nanoseconds could be achieved. Other state of the art solutions achieve a precision of 2 microseconds.

6.2 Outlook

The work has shown that the process of clock synchronization is influenced by many factors. It has been shown that improvements can be made by understanding the sources of errors. Especially the analysis of the delays has shown where simple improvements can be made. Filtering the incoming data has proven to be particularly helpful. But to achieve really big improvements, there is no way around hardware timestamping. This measure alone made it possible to achieve the goals in terms of precision.

With regard to the solution with software timestamping, however, there is still a lot that can be done here. Here, the possibilities were severely limited by the given system architecture. A pure software solution would have the great advantage of easy portability, since in this case no FPGA is necessary. There is still potential for further work in this area, especially with regard to future CAN extensions such as CAN XL.

AUTOSAR time synchronization over CAN was selected as the protocol here. Since this is a very simple protocol and its main task is to send timestamps, it could be replaced by another protocol with a similar structure. If compatibility to standard CAN is not a criterion, PTP could be implemented directly based on e.g. CAN XL. This would have the advantage that a delay measurement is already part of the protocol.

Instead of relying on statistical filtering, one could also choose a completely different approach. One possibility would be a Kalman filter. This requires an underlying model. This would give the possibility to model the errors and properties of the system exactly. Especially for pure software solutions this would be very interesting, because there is still potential for improvement.

Further work would be necessary to evaluate these possibilities.
List of Figures

2.1 Clock generation in computer systems ........................................ 4
2.2 Rate error ................................................................................... 5
2.3 Timestamping error ........................................................................ 6
2.4 Synchronization condition .............................................................. 7
2.5 Communication model ...................................................................... 8
2.6 Delay measurement .......................................................................... 11
2.7 CAN data frame ................................................................................ 12
2.8 PTP message exchange with peer-to-peer delay measurement .......... 16
2.9 Autosar time synchronization over CAN message exchange .......... 18

3.1 EBX200 platform ............................................................................ 26
3.2 Timestamping points of MCAN and PTP ........................................ 27
3.3 CAN frame detection unit ................................................................. 28
3.4 Timestamping point of frame detection unit ..................................... 29
3.5 Position of timestamps in CAN frame ................................................ 29
3.6 Elektrobit highspeed capture and replay protocol header ................. 32
3.7 Overview of timing related features .................................................. 34
3.8 Delay model .................................................................................... 35
3.9 Software timestamping delays .......................................................... 36
3.10 Coinciding scheduling periods ......................................................... 37
3.11 Asynchronous scheduler case 2 ....................................................... 38
3.12 Asynchronous scheduler case 3 ....................................................... 38
3.13 Case 1: Timestamps are created at the same time .......................... 39
3.14 Case 2: Slave creates the timestamp first ........................................ 40
3.15 Case 3: Master creates the timestamp first ......................................... 41
3.16 Delay of hardware timestamping without MCAN delay measurement ........ 42
3.17 Delay of hardware timestamping with MCAN delay measurement ........... 43

4.1 Timestamps used for synchronization ................................................. 46
4.2 Offset correction structure ................................................................. 47
4.3 Offset adjustment .............................................................................. 49
4.4 Temporary rate adjustment ................................................................ 50
4.5 Rate correction structure .................................................................... 51
4.6 Rate correction with filter .................................................................. 54
4.7 Moving average filter ......................................................................... 55
4.8 Exponentially weighted average .......................................................... 56
4.9 Moving median filter .......................................................................... 57
4.10 Relation of filter width and coefficient ............................................... 59
4.11 CAN synchronization with delay measurement .................................... 63

5.1 Measurement via timer GPIO .............................................................. 65
5.2 Measurement via DETI ....................................................................... 66
5.3 Timer GPIO measurement of software solution .................................... 67
5.4 DETI measurement of software solution ............................................ 68
5.5 Timer GPIO measurement of software solution with filtering ............... 69
5.6 DETI measurement of software solution with filtering ........................ 70
5.7 DETI measurement of improved software solution .............................. 71
5.8 DETI measurement of improved software solution with filtering .......... 72
5.9 Timer GPIO measurement of hardware solution ................................... 73
5.10 DETI measurement of hardware solution ......................................... 74
5.11 Timer GPIO measurement of hardware solution with filtering .......... 75
5.12 DETI measurement of hardware solution with filtering .................... 76
5.13 Timer GPIO measurement of hardware solution with filtering and delay compensation ................................................................. 77
5.14 DETI measurement of hardware solution with filtering and delay compensation and bus load ................................................................. 78
References


[AL04a] Allan, Jeff; Lee, Dongik: Fault-Tolerant Clock Synchronization with Microsecond-Precision for CAN Networked Systems, 2004

[AL04b] Allan, Jeff; Lee, Dongik: Fault-Tolerant Clock Synchronization with Microsecond-Precision for CAN Networked Systems, 2004

[ASC17] Specification of Time Synchronization over CAN. In: AUTOSAR CPRelease 4.3.1 (2017), S. 1–70


[BKSG17] Breaban, Gabriela; Koedam, Martijn; Stuijk, Sander; Goossens, Kees: Time Synchronization for an Emulated CAN Device on a Multi-Processor System on Chip. In: Microprocessors and Microsystems 52 (2017), 05


[CP09] Carvalho, Fabiano C.; Pereira, Carlos E.: A runtime stability analysis of clock synchronization precision on a time-triggered bus prototype. 20 (2009), 03, S. 45 – 52. – ISSN 0103-1759

[DB16] Daigmorte, Hugo; Boyer, Marc: Traversal time for weakly synchronized CAN bus, 2016, S. 35–44

[DBM17] Daigmorte, Hugo; Boyer, Marc; Migge, Jörn: Reducing CAN latencies by use of weak synchronization between stations, 2017


[FMD+01] Führer, Thomas; Müller, Bernd; Dieterle, Werner; Hartwich, Florian; Hugel, Robert; Walthier, Michael; Bosch, Robert: Time Triggered Communication on CAN (Time Triggered CAN TTCAN). (2001), 01


[GS94] Gergeleit, Martin; Streich, H.: Implementing a distributed high-resolution real-time clock using the CAN-bus, 1994


[HHV02] Horauer, Martin; Höller, Roland; Voice, Le: Integration of high accurate Clock Synchronization into Ethernet-based Distributed Systems. (2002), 01

[HMF+02] Hartwich, Florian; Müller, Bernd; Führer, Thomas; Hugel, Robert; Gmbh, Robert: Timing in the TTCAN Network. (2002), 01


[MVLF08] Marto, Pau; Velasco, Manel; Lozoya, Camilo; Fuertes, Josep: Clock Synchronization for Networked Control Systems Using Low-Cost Microcontrollers. (2008), 08

[RFSS17] Rinaldi, Stefano; Ferrari, P.; Sisinni, Emiliano; Sauter, Thilo: Network
REFERENCES

Synchronization: An introduction. 2017


