CLOCK SYNCHRONIZATION AND WEAK TDMA FOR CAN FD: IMPLEMENTATION AND EVALUATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

İSMET ONUR DEMIREL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

DECEMBER 2022

Approval of the thesis:

CLOCK SYNCHRONIZATION AND WEAK TDMA FOR CAN FD: IMPLEMENTATION AND EVALUATION

submitted by **İSMET ONUR DEMIREL** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İlkay Ulusoy
Head of Department, **Electrical and Electronics Engineering** _____

Prof. Dr. Ece Güran Schmidt
Supervisor, **Electrical and Electronics Engineering, METU** _____

Prof. Dr. Klaus Werner Schmidt
Co-supervisor, **Electrical and Electronics Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering, METU _____

Prof. Dr. Ece Güran Schmidt
Electrical and Electronics Engineering, METU _____

Prof. Dr. İlkay Ulusoy
Electrical and Electronics Engineering, METU _____

Assoc. Prof. Dr. Mustafa Mert Ankaralı
Electrical and Electronics Engineering, METU _____

Assist. Prof. Dr. Ulaş Beldek
Mechatronics Engineering, Çankaya University _____

Date: 27.12.2022

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: İsmet Onur Demirel

Signature :

ABSTRACT

CLOCK SYNCHRONIZATION AND WEAK TDMA FOR CAN FD: IMPLEMENTATION AND EVALUATION

Demirel, İsmet Onur
M.S., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Ece Güran Schmidt
Co-Supervisor: Prof. Dr. Klaus Werner Schmidt

December 2022, 67 pages

The Controller Area Network (CAN) is the most widespread in-vehicle communication protocol. Although CAN has been used for many decades, there are very recent software-based enhancements that enable accurate clock synchronization and time-slotted medium access on CAN, denoted as weak time division multiple access (WTDMA), for deterministic bus access. In addition, there are recent updates to the CAN hardware such as CAN with flexible data rate (CAN FD) in order to both increase the available bit rate and the length of CAN message frames.

This thesis takes into account that the mentioned software-based improvements have been proposed and evaluated for the standard CAN protocol but not for CAN FD. That is, the thesis first realizes different clock synchronization methods for CAN FD on state-of-the-art microcontroller evaluation boards. Making use of these clock synchronization methods, the thesis further implements WTDMA and confirms its correct operation for CAN FD. As an important feature, all the implementations are done in software and are hence compatible to both CAN and CAN FD standard.

Keywords: Controller Area Network, CAN with Flexible Data Rate, Clock Synchronization, Weak TDMA

ÖZ

CAN FD İÇİN ZAMAN SENKRONİZASYONU VE ZAYIF TDMA: GERÇEKLEME VE DEĞERLENDİRME

Demirel, İsmet Onur
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Ece Güran Schmidt
Ortak Tez Yöneticisi: Prof. Dr. Klaus Werner Schmidt

Aralık 2022 , 67 sayfa

Kontrolör Alan Ağı (CAN) araç içi iletişim protokolleri arasında en yaygın olanıdır. On yıllardır kullanılıyor olmasına rağmen, güncel olarak CAN üzerinde yazılım tabanlı kesin saat senkronizasyonunu sağlamak ve deterministik ortam erişimi için zaman bölmeli ortam erişimi bir başka deyişle zayıf zaman bölmeli çoklu erişim (WTDMA) üzerine iyileştirmeler yapılmaktadır. Ayrıca, CAN üzerinde veri hızını ve CAN mesaj çerçevesinin uzunluğunu artırmak için donanımsal güncellemeler, örneğin CAN Esnek Verihızı (CAN FD), yapılmaktadır.

Bu tezde, bahsedilen yazılım bazlı geliştirmelerin standart CAN için sunulduğu ve değerlendirildiği fakat CAN FD için yapılmadığı göz önünde bulundurulmuştur. Bu tez, ilk kez, farklı saat senkronizasyonu methodlarını CAN FD için teknoloji harikası mikrokontrolcü işlemci kartlarında gerçeklemiştir. Bu saat senkronizasyonu methodları ile WTDMA uygulamasının CAN FD için doğru çalıştığını onaylar. Bu çalışmadaki önemli bir özellik ise bu uygulamaların tamamen yazılım üzerinde yapıldığı ve dolayısıyla hem CAN hem de CAN FD standartlarına uyumlu olduğudur.

Anahtar Kelimeler: Kontrolör Alan Ağı, CAN Esnek Verihızı, Saat Senkronizasyonu, Zayıf TDMA

To my family and fiancé

ACKNOWLEDGMENTS

Foremost, I would like to express my deepest gratitude to my advisors Prof. Dr. Şenan Ece Schmidt and Prof. Dr. Klaus Werner Schmidt for their trust, endless support, encouragement, and guidance throughout the studies. Additional to my advisors, I would like to thank Murat Akpınar for sharing his knowledge and experience, and helping me in my studies.

I also thank TUBITAK (Scientific and Technological Research Council of Turkey) for funding the studies conducted in this thesis [Project Code 119E277].

I want to thank my friends Güray, Sahra, Fatih, Hakan and Kaan Furkan for standing with me during the thesis work. I would also thank to my family for supporting me all the time.

Finally, I would like to present my gratitude to my fiancé for backing me up whenever I needed.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ACK	Acknowledgement
CAN	Controller Area Network
CAN FD	CAN Flexible Datarate
CAN-DS	CAN with Determinism and Synchronization
CRC	Cyclic Redundancy Check
CO	Clock Oscillator
CS	Clock Synchronization
DLC	Data Length Code
EoF	End of Frame
FUP	Follow Up
GPIO	General Purpose Input Output
HR	Hardware Register
HP	Hyper Period
IA	Initial Accuracy
ID	Identification
IDE	Integrated Development Environment
IFS	Inter Frame Spacing
LC	Local Clock
LTI	Long Term Instability
lcm	least common multiplier
MCU	MicroController Unit
ppm	parts per million
QoS	Quality of Service
RC	Reference Clock

RM	Reference Message
RT	Response Time
RTR	Remote Transmission Request
SoF	Start of Frame
SW-CS	Software-Based Clock Synchronization
SYNC	Synchronization
TDMA	Time Division Multiple Access
TM	Time Master
TR	Trigger
TS	Timestamp
TTCAN	Time Triggered CAN
WCRT	Worst Case Response Time

CHAPTER 1

INTRODUCTION

The Controller Area Network (CAN) [1] is the most prominent in-vehicle communication bus in modern cars [2, 3, 4, 5]. Specifically, CAN is still indispensable for safety critical components such as engine control, transmission control, braking, steering and suspension control that require high real-time quality of service (QoS) [4, 6, 7]. After the first introduction in the mid 1980s, the advancement of the legacy CAN protocol is yet ongoing with extensions such as CAN with Flexible Data-rate (CAN FD) [8] in 2012 and CAN XL [9] starting from 2018. In particular, CAN XL will offer data rates above 10 Mbps and payloads of up to 2048 bytes [6, 9] to meet the stringent demand of modern applications. Due to the continuing importance of CAN, the recent literature is highly interested in possible improvements for CAN such as advanced clock synchronization (CS) [10, 11, 6] and security methods [4, 2, 5].

Real-time messages on CAN need to be received before their specified deadline [12, 13, 14]. To evaluate this deadline constraint, the conventional schedulability analysis proposed in [15] and its revised version [16] have been used in the automotive industry [17]. In this analysis, the worst-case response times (WCRTs) of CAN messages are computed by considering the unlikely scenario, where all CAN messages are ready for transmission at the same time. Here, it is claimed that bus utilization up to 80% can be reached with a suitable priority assignment policy [18]. Nevertheless, it is not possible to freely change the pre-assigned priorities of CAN messages in industrial applications [19, 20] such that approaches for increasing the bus utilisation that only benefit from the optimal priority assignments are not practical. Furthermore, such priority assignment methods provide a lower bus utilization in case of tighter deadlines.

It is shown that assigning offsets to CAN messages of individual nodes increases the efficiency of the bandwidth usage by spreading the message load of each node over time [21, 19]. Even though CAN messages from different nodes may still be ready for transmission simultaneously, a major performance improvement is achieved with the usage of offsets in terms of WCRTs even without applying CS [19]. Additionally, the bounded phases concept, which benefits from CS in the order of ms has been introduced in [22]. It is shown in [19] that the usage of offsets with a CS accuracy of 1 ms allows reaching an average bus load of 83% if the deadlines are assumed to be equal to message periods. It has to be noted that the given bus load is computed with the breakdown utilisation notion in [18] that facilitates to reach higher bus utilisation with higher deadline requirements. Even though the bounded phase approach with offset assignments seems to facilitate CAN messages meeting their deadlines, its usability in case of tight deadline requirements is questionable.

In addition to software-based approaches such as the offset assignment and the bounded phase concept, Time Triggered CAN (TTCAN) [23] has been proposed to enable deterministic bus access at the expense of the modification of the CAN protocol and its underlying hardware. A bus usage close to 100% seems achievable in TTCAN since a global clock among the nodes makes it possible to schedule all CAN messages in a deterministic way [24, 19]. However, to date TTCAN has not been used in production cars [25] to the best of our knowledge due to the lack of compatibility with the existing CAN protocol and hardware.

Apart from TTCAN for in-vehicle networks, Time Division Multiple Access (TDMA) is employed in many communication systems to provide collision-free transmission with bounded access delay [26]. In TDMA, time is divided into isolated windows and only one node is allowed to transmit in a specific window. Moreover, guard times are introduced to avoid any interference between windows of different nodes [27], which leads to a reduction in the bus utilization.

In addition to the described previous methods, the very recent work in [28] makes two important observations. Although there is progress in CS for CAN, there are various possible methods for improving the clock accuracy. Such improvements pave the way for the deterministic medium access on CAN, which is highly desired to provide

efficient bandwidth utilization and deterministic message response times [19, 26, 27]. Second, it holds that CAN is a non-preemptive bus and possible message collisions are managed by the CAN arbitration mechanism without a performance penalty. That is, when realizing window-based access on CAN it is not required to temporarily isolate the windows of different nodes since possible overlaps will be resolved by the usage of Carrier Sense Multiple Access/Collision Resolution (CSMA/CR) and the non-preemptive message transmission on CAN. That is, there are no very stringent CS requirements when realizing TDMA over CAN. Accordingly, [28] proposes a novel Weak TDMA (WTDMA) model for CAN, where time slots can temporarily overlap, different from the classical TDMA. Specifically, WTDMA can operate even with a moderate clock accuracy among the nodes that can even be provided with any of the existing software-based CS (SW-CS) methods for CAN such as [29, 11]. Nevertheless, it has to be noted that WTDMA has only been realized and evaluated for the standard CAN protocol but not for the more advanced CAN FD.

The main contribution of this thesis is the realization of clock synchronization algorithms and WTDMA for CAN FD. The thesis first implements two different offset correction algorithms on recent microcontroller evaluation boards that support CAN FD. Gergeleit's method [30] is implemented as an efficient method that requires the minimum number of additional messages on the CAN bus. Furthermore, the AUTOSAR method is realized to achieve a better clock accuracy. Since offset correction methods cannot ensure a sufficient clock accuracy, an additional drift correction method is developed and implemented. As a result, a clock accuracy in the order of few microseconds is achieved. The thesis further implements WTDMA for CAN FD and confirms its correct operation for different parameter settings.

The remainder of the thesis is as follows. Chapter 2 gives the required background information on CAN and clock synchronization. The detailed concepts and implementation steps are described in Chapter 3 and evaluated in Chapter 4. Finally, Chapter 5 gives conclusions.

CHAPTER 2

BACKGROUND INFORMATION

In this chapter, the background information needed throughout this thesis is introduced. First, the basic information about the CAN protocol and its message formats is given. Afterwards, CAN FD (CAN with flexible data rate) is explained which is the extension of CAN protocol that enables higher bus speed and larger message payloads. Moreover, clock synchronization for CAN is explained since it constitutes the basic concept used in this thesis. Before explaining used offset correction methods, local clock and clock drift concepts are described to be more clear. Gergeleit's and AUTOSAR methods which are the mostly used offset correction methods are explained. In addition, drift correction is discussed as a means to improve the clock synchronization quality. In the last part, the adaption of time division multiple access (TDMA) to CAN in the form of weak TDMA is explained.

2.1 General Background on CAN

CAN is an asynchronous, multi-master, serial data bus that was designed by Robert Bosch GmbH in 1983 and was standardized in 1993. CAN realizes a priority-based non-preemptive bus arbitration, whereby the present message with the highest priority gains access to an empty bus. Since it was developed with this purpose, it is highly applicable for real-time control operations in vehicles. Apart from the automotive industry, CAN is applicable in different industrial safety critical applications that need communication between multiple devices within certain time limits. It provides a simple network by reducing wiring and allowing multiple microcontrollers to communicate on a single bus.

CAN protocol is defined with both of it's hardware and software specifications. Hardware specifications contains Physical Layer control. The messages are transferred by two wires, and bits are sent by differential signalling. These are not in the scope of this thesis, however, these are the main reasons that make CAN simple, cheap and less error prone in the application areas with electrical noise.

Physical Layer control of CAN is defined with the first introduction of CAN, and used as the same by now. However, software specifications and it's extensions are improved as time passes. Standard CAN has a 1 Mbps bus speed limit, and a maximum message payload of 8 Bytes. In order to exceed these limits, CAN FD in 2012 [31] and CAN XL in 2018 [9] are presented in the literature. CAN Flexible Data-rate offers up to 10 Mbps in the payload and CAN XL offers up to 2048 bytes payload in order to extend standard CAN limitations.

2.2 CAN Messages

CAN messages are sent in frames to control communication. There are four different types of frame in CAN. These are data frame, remote frame, error frame and overload frame. The data frame is the standard frame for sending data. Remote frame is used to ask for transmission of a message with the same ID from a different node. Error frames can be transmitted by any node when an error is detected in message transfer. The overload frame is used for providing time delay between remote and data frames.

In a standard CAN data frame, there exist seven different fields as shown in Figure 2.1. These are start of frame field (SoF), arbitration field, control field, data field, cyclic redundancy check (CRC) field, acknowledgement (ACK) field, and end of frame (EoF) field. A node starts sending a message by setting the start of frame bit to dominant if there is no message on the bus. Then ID field is sent to bus, where the arbitration starts in case of any other nodes are sending their messages to the bus. By the end of ID field, the arbitration ends and the highest priority message continues sending its message.

CAN also supports an extended format where there is 29 bit ID as shown in Figure 2.2 instead of 11 in the standard format. In the extended format, the arbitration ends with

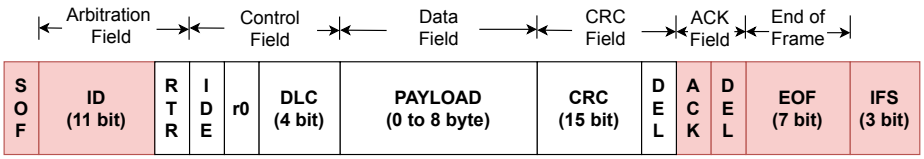


Figure 2.1: CAN frame - standard format.

the extension of ID. Afterwards, there is a 6 bit control field that contains the data length code (DLC) which states the total size of payload that can be varying between 0 to 8 bytes. The 16 bit CRC field is consisting of 15 bit CRC coding and a recessive CRC delimiter bit. This CRC sequence is the security check for all receiving nodes. If any invalid receiving node gets invalid CRC coding during the transmission of the message, it sends error frame. At the end of a data frame before the end of frame field, there is ACK field consisting of ACK and ACK delimiter bits. Sender node sets the ACK bit to recessive and receiving nodes send a dominant bit for the first bit of the ACK field. The end of frame field is made up of seven consecutive recessive bits. There should be also at least three recessive bits as inter frame spacing between each frames.

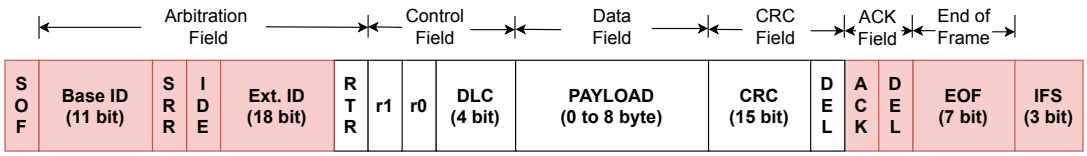


Figure 2.2: CAN frame - extended format.

Both of the above formats of CAN data frame have 7 fields. Except from the data field, the other 6 field have constant length in bytes. Therefore, the total length of the CAN frame, is sum of 6 constant length and a variable size data field length. This total length is also effected by the bit stuffing applied by the CAN protocol itself, which is a bit of opposite value is inserted into the message after five consecutive same value bits [1]. By considering these information, the frame bit length varies between a minimum value f_{min} (determined by the frame header and payload) and a

maximum value f_{max} (determined by the maximum bit stuffing) as follows [16]:

$$f_{min} = g + 8 \cdot l + 13 \tag{2.1}$$

$$f_{max} = g + 8 \cdot l + 13 + \left\lfloor \frac{g + 8 \cdot l - 1}{4} \right\rfloor, \tag{2.2}$$

whereby $g = 34$ for the standard format (11-bit identifiers) and $g = 54$ for the extended format (29-bit identifiers) of CAN. l is varying between 0 to 8 as the payload size. The corresponding values for the possible standard format CAN frame bit lengths are shown in Table 2.1.

l	0	1	2	3	4	5	6	7	8
f_{min}	47	55	63	71	79	87	95	103	111
f_{max}	55	65	75	85	95	105	115	125	135

Table 2.1: Minimum and maximum of the frame bit length depending on l.

Moreover, we call the bit rate B and the corresponding bit time $\tau_{\text{bit}} = 1/B$. Using τ_{bit}, in terms of time duration, the bounds of the message length of M is calculated as $L_{min} = f_{min} \cdot \tau_{\text{bit}}$ and $L_{max} = f_{max} \cdot \tau_{\text{bit}}$, respectively.

2.3 Standard CAN Protocol

CAN communication is based on broadcasting fixed type of messages. The message has payload, which is the meaningful part of the message, with the size of 0 to 8 bytes range. Bit rate is limited between 10 kbps and 1 Mbps for the standard CAN protocol. However, all nodes in the system must have the same speed for the successful and durable operation.

CAN protocol is more convenient to be used in multiple node systems by broadcasting messages, rather than two node communication each other. All nodes can sense the medium and any of them can start sending a message if there is no other ongoing message. The messages do not have a receiver ID, instead they have message ID which must be unique for proper operation. The message on the bus is received by all other nodes simultaneously. Nodes can accept the message or ignore it by

differentiating from it's message ID. Moreover, each node sends an acknowledgement after each message to the transmitter node in order to inform that transmission is successful.

The nodes are receiving messages by sensing the bus level. The voltage level of the bus is not concerned here, on the other hand some more information about the bus level should be given in order to explain how CAN is a non-preemptive communication protocol. The bus level can be either logic '1' or '0'. For CAN bus, logic '0' bit is called dominant bit and logic '1' bit is called recessive bit. The nodes are sensing the medium before sending message, and wait until the bus is empty. When the bus become empty, more than one node can start transmission at the same time. In order to prevent collision of simultaneously sent two different messages, the bus level is determined by a logic AND operation such that bus value is dominant if at least one of the bits is '0' among the messages. Two messages can not be transferred in the bus at the same time, therefore the message priority is considered at this point. The arbitration between multiple messages is solved by bit-wise arbitration based on the message priorities in case of multiple nodes started transmitting at the same time. During this arbitration, each transmitter node senses the bus level and checks whether it is the sent bit or not. Until a level inequality, node continues sending the message and sensing the medium. In the inequality case, it means a message with higher priority (smaller ID value) is sent at the same time by a different CAN node. Hence, the sender stops sending the message and gives the bus access to the higher priority message. This arbitration sequence is happening in the ID sending time, after a successful arbitration other transmitter nodes waits for the bus being empty again after the message is sent. This behaviour makes CAN protocol non-preemptive and solves the arbitration without performance degradation since no data loss or bandwidth loss happens.

2.4 CAN FD

It has to be noted that the standard CAN Bus does not meet the data rate requirements of contemporary in-vehicle communication anymore. However, CAN is a trusted protocol being used for many years in countless applications in the automotive industry.

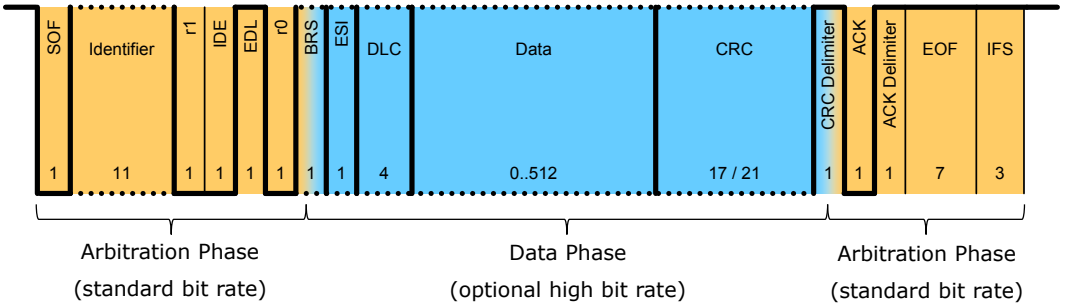
For these reasons, CAN FD (CAN with Flexible Data Rate) which both offers much higher bandwidth than CAN and backward compatibility with CAN is developed by Bosch [31, 32].

CAN FD operates at two different bit rates within a message frame. It has the same bit rate and the arbitration method as CAN but it switches to a higher bit rate during the data phase. CAN FD payload is up to 64 Bytes. Therefore at a given bus load, the overhead of the frame decreases down to 15% [33] and theoretical net bit rates about 5 Mbps are possible [32]. Furthermore, the arbitration phase baud rate limits the overall baud rate of the frame. For example, a frame with 64 bytes payload, 11 bit standard ID, 1 Mbps of arbitration phase baudrate, 8 Mbps of dataphase baud rate has net bit rate about 5.9 Mbps [34]. Furthermore, the increase of baud rate is also beneficial for higher layer software protocols. [35] assesses the effectiveness and performance of CAN FD with respect to CAN bus in agricultural systems using higher layer protocols like J1939 and ISOBUS.

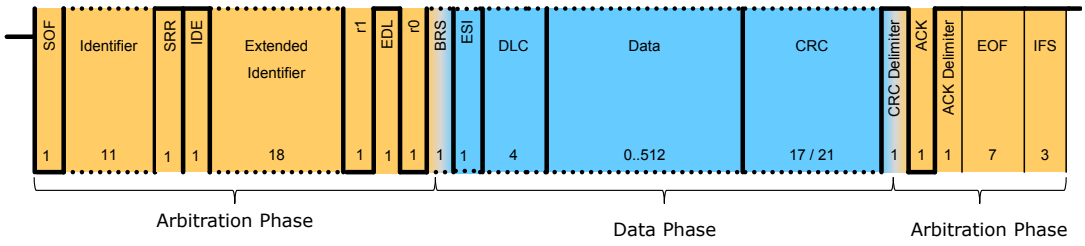
Car manufacturers begin to adapt CAN FD in their system design. Toyota, Denso, and Renesas cooperate for autonomous driving system developments. Renesas contributes with micro controllers and System on Chip (SoC) devices featuring CAN FD [36]. According to [37], Mercedes considers introducing CAN FD in their S-class series cars. There are some works to adopt CAN FD to real network systems and CAN FD is considered to be used in the same network with CAN [38], [39].

According to [40], CAN FD data phase bit rates up to 2 Mbit/s will be used in the first CAN FD systems. The network topology will be like star or hybrid. Later generation CAN FD systems will increase the data rate up to 5 Mbit/s. CAN FD frames can be divided into three parts, which are arbitration phase, data phase and arbitration phase again as can be seen in Fig.2.3. The bit rate switches to higher rate only during data phase and switches back to its old rate when the data phase is over.

CAN FD frame format differs from CAN frame format in terms of payload length, Data Length Code (DLC) and CRC computation method. Therefore, some hardware changes are required in the controllers. If the payload size is kept as 8 Bytes as in standard CAN messages, there is no need for any software changes [41]. CAN FD supports payload size of up to 64 Bytes. In such implementations software changes



(a) CAN FD Base Frame



(b) CAN FD Extended Frame

Figure 2.3: CAN FD Frames

are required. The cost to implement CAN FD is very similar to CAN implementation costs [31].

The frames with the base ID have 11 bits ID representation while the frames with the extended ID has 29 bits ID representation. Furthermore, some control bit values in the frame change according to the ID type. CAN FD data frames for Base and Extended ID can be seen in Fig.2.3a and in Fig.2.3b.

Since the CAN-FD frames have differences with standard CAN frames, their bit length is calculated differently compared to the equations in 2.1 and 2.2. Moreover, the CAN-FD frame has different rates for different fields which makes bit length calculation insufficient in its own. Therefore, it is better to calculate time duration of the message length. Bit stuffing is still valid for CAN-FD frames, hence message length

calculations for the maximum length is as follows [42]:

$$L_{min} = (g + 18) \cdot \tau_{\mathrm{a}} + (27 + 8 \cdot l) \cdot \tau_{\mathrm{d}} \tag{2.3}$$

$$L_{max} = (g + \left\lfloor \frac{g+2}{4} \right\rfloor + 12) \cdot \tau_{\mathrm{a}} + (27 + 8 \cdot l + \left\lfloor \frac{8 \cdot l + 6}{4} \right\rfloor) \cdot \tau_{\mathrm{d}}, \tag{2.4}$$

whereby g is equal to identifier size, that is either 11 or 29 bit, depending on the CAN message format. l stands for the payload size and varies between 0 and 64 in CAN-FD. CRC field is 17 bits plus 4 fixed stuff bits for payload sizes up to 16 bytes, and 21 bits plus 5 fixed stuff bits for payload sizes larger than 16 bytes. In case of bit stuffing, after slowing down to arbitration speed, the ACK, EOF and IFS fields do not apply bit stuffing [42].

2.5 Clock Drift

The local clock of a node, called LC in this thesis, is briefly explained as timer module features of a microcontroller unit. The main components of LC are the clock source and the clock value. Clock source is mostly a crystal or sometimes internal clock generators, and they are called clock oscillator (CO). In every periodic tick from CO, a hardware register (HR) is incremented, which forms the clock value [43, 44]. The clock value has two properties, resolution and maximum value. Resolution of the clock states the time amount between each clock value increment. This time amount depends on two things, the nominal frequency $f_{\mathrm{CO},N} = 1/T_{\mathrm{CO},N}$ of CO and the number of CO ticks $n_{\mathrm{HR},N}$ for each increment. Using these factors, one increment in a node N's local clock value equals to a time duration $T_N = n_{\mathrm{HR},N} \cdot T_{\mathrm{CO},N} = n_{\mathrm{HR},N}/f_{\mathrm{CO},N}$. The other property, maximum value of clock, depends on the storage of clock value variable. The width of the HR determines the maximum value of the clock. The resolution and the maximum value determine the maximum time amount that local clock can evaluate. After that time, the operation varies with respect to software decision. The common use case is being aware of maximum value and counting again starting from zero.

After giving brief information about local clock, it is time to explain clock drift. Clock drift occurs by having different time values in LC for different CAN nodes. This difference is caused by the fact that each CO in CAN nodes have different rate

in providing periodical ticks [43]. This rate change occurs due to a deviation in actual frequency of CO. This difference of actual frequency and ideal frequency arise from short term, long term and environmental frequency instability effects [45, 46]. This difference is measured in ppm (parts per million). The fabrication errors causes frequency instability in the order of 50 ppm, aging per year and temperature variations (between -40 to 125 °C) causes 5 ppm and up to 150 ppm accordingly [47]. Knowing these, the actual frequency f_N of node N's CO is dependent to time and can be expressed in the form

$$f_N(t) = f_{\mathrm{CO},N} \cdot (1 + k_{\mathrm{IA},N} + k_{\mathrm{LTI},N} + k_{\mathrm{EI},N}(t)). \tag{2.5}$$

Here, $k_{\mathrm{IA},N}$ is the initial accuracy (IA) based on fabrication errors, $k_{\mathrm{LTI},N}$ is the long term frequency instability (LTI) due to aging and $k_{\mathrm{EI},N}(t)$ represents the time-varying environmental instability (EI) depending on temperature, pressure, humidity and noise from the voltage supply [43, 48, 49]. Then, the local time $c_N(t)$ of node N's LC is determined by

$$c_N(t) = N_N(t) \cdot T_N = \Big\lfloor \int_0^t \frac{f_N(\tau)}{n_{\mathrm{HR},N}} d\tau \Big\rfloor \cdot T_N, \tag{2.6}$$

whereby $\lfloor \bullet \rfloor$ is the floor operation. As stated before, the LC of each CAN node diverges from real time due to these frequency instability reasons.

In order to mention about a time difference, we define the clock difference of node N with respect to a reference clock (RC) which is accepted to equal to the real time as

$$\Delta c_N(t) = c_N(t) - c_{\mathrm{RC}}(t). \tag{2.7}$$

When we consider that real time is given by the LC value c_{TM} of a given master node in the experiments of this thesis, it holds that $c_{\mathrm{RC}}(t) = c_{\mathrm{TM}}(t)$. Then, a node N has the clock offset $\Delta N_N(t) = N_N(t) - \left\lfloor \frac{c_{\mathrm{TM}}(t)}{T_{\mathrm{N}}} \right\rfloor$ in terms of clock value. Next, the accuracy of the node N's accuracy is described as the maximum value of clock difference $\max_t |\Delta c_N(t)|$. As a result of these calculations, a node N's clock drift d_N is the rate of change of $\Delta c_N(t)$ and can be approximated for small values of τ as

$$d_N(t) = \frac{d}{dt}(\Delta c_N(t)) \approx \frac{\Delta c_N(t) - \Delta c_N(t-\tau)}{\tau}. \tag{2.8}$$

2.6 Clock Synchronization

This section outlines the concept of clock synchronization on CAN. We consider a time master (TM) node, which is assumed to have a perfect local clock and a generic slave node S_y, whose local clock has to be synchronized with the TM. Writing e_{S_y} for the local clock difference of S_y and TM, Fig. 2.4 (a) shows the behavior if no clock synchronization is applied. That is, e_{S_y} changes linearly due to the unavoidable oscillator drifts of S_y and TM [43].

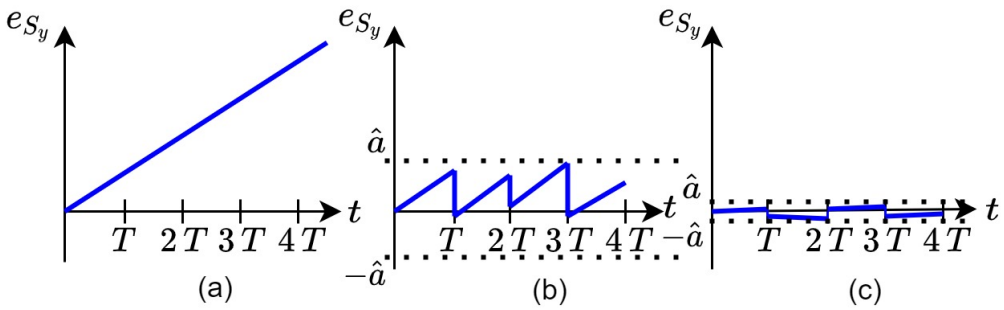


Figure 2.4: (a) Clock drift; (b) Offset correction; (c) Drift correction.

Clock synchronization on CAN involves *offset correction* and *drift correction*. Both techniques are commonly based on timestamps that are taken simultaneously by each node and the TM transmits its timestamp with a period T on the CAN bus. Any slave node S_y periodically applies offset correction [30, 29, 10] by adjusting its local clock based on the difference of its own timestamp and the received timestamp from the TM as shown in Fig. 2.4 (b). That is, after offset correction at $T, 2\,T, \ldots$, e_{S_y} is close to zero, whereas the local clock of S_y drifts until the next offset correction. In any case, $|e_{S_y}|$ is bounded by a maximum value $\hat{a}$ that quantifies the *clock accuracy* of the clock synchronization method. The clock drift between offset corrections can be further reduced by applying drift correction based on drift estimates from the timestamps [50, 51, 6, 52] as illustrated in Fig. 2.4 (c).

It has to be emphasized that offset correction directly uses timestamps and drift correction is performed based on drift estimates that are computed from timestamps. That is, the clock accuracy of CS methods strongly depends on the timestamp accuracy, which is an important criterion for the implementation of CS methods on CAN.

CHAPTER 3

CONTRIBUTION

This chapter describes the methods implemented in the scope of this thesis. Section 3.1 explains the realization of timestamping and Section 3.2 provides details about the application of offset correction. Drift correction is implemented in Section 3.3 and realization of time division multiple access (TDMA) for CAN is discussed in Section 3.4.

3.1 Timestamping

3.1.1 Timestamping Background

Timestamping on CAN can be characterized by two main components. First, the TS process is initiated with a trigger (TR) signal at the TR instant on each CAN node, which occurs during a periodic reference message (RM) sent by the TM. Second, the TR signal is detected and a copy of the current value of the local clock is taken by a function that is denoted as the TS service. The actual time instant when the timestamp is taken is indicated by a TS signal after the completion of the TS service. The basic setting is illustrated in Fig. 3.1 for an arbitrary k-th TS instant of the TM and a slave node S_y.

Here, $t^{\mathrm{TR}}_{\mathrm{TM},k}$ and $t^{\mathrm{TR}}_{S_y,k}$ are the respective TR instants, $d^{\mathrm{SV}}_{\mathrm{TM},k}$ and $d^{\mathrm{SV}}_{S_y,k}$ show the respective time durations of the TS service and $t^{\mathrm{TS}}_{\mathrm{TM},k}$ and $t^{\mathrm{TS}}_{S_y,k}$ represent the times of the supposedly synchronous TS signals for the TM and S_y.

The TS quality of a slave node S_y is characterized by the distribution of the differences between $t^{\mathrm{TS}}_{\mathrm{TM},k}$ and $t^{\mathrm{TS}}_{S_y,k}$. Accordingly, the TS quality directly depends on the quality

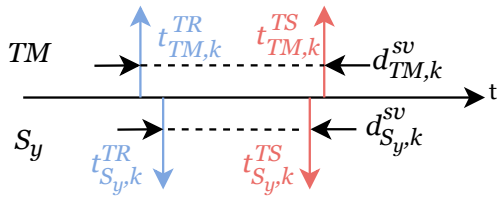


Figure 3.1: The components of TS on CAN.

of the TR instant and TS service.

It has to be noted that software implementations commonly use the end of the reference message (EoM) [30, 50, 51, 29, 50] as specified in the CAN in Automation (CiA) standard 603 [53] as the TR instant. Since this TR instant is directly determined by the internal bit timing mechanisms of CAN, the TR instant quality depends on the CAN protocol itself. Differently, the TS service quality depends on its implementation. An implementation in hardware is expected to provide a constant time of the TS service and will hence perform better than a software implementation with potentially varying delays. Accordingly, when implementing timestamping in software, much care has to be taken in order to achieve deterministic execution times of the TS service.

3.1.2 Timestamping Implementation

Timestamping on the TM is implemented in the form of a timer (TC4) that creates an interrupt with the RM period T. In addition, a second timer (TC0) holds the local time of the TM. The timers are configured as shown below.

```
TC4_TimerCallbackRegister(TC4_Callback_InterruptHandler,
    (uintptr_t)NULL);
TC4_TimerStart();
TC0_TimerStart();
```

Whenever the timer for the RM expires, the callback function "TC4_Callback_InterruptHandler" is called to take a timestamp.

```
void TC4_Callback_InterruptHandler(TC_TIMER_STATUS status,
    uintptr_t context){
    timestampValue = TC0_Timer32bitCounterGet();
}
```

A timestamp on the slave is taken whenever a RM is received. RM message properties change between Gergeleit's method and the AUTOSAR methods. Still, the implementation of timestamping is similar. To this end, it is required to configure the receive message frame and the callback register for message reception. This example is from the AUTOSAR implementation of a slave. Here, the timestamp is taken after the follow-up (FUP) message. FUP is filtered and saved into the RX buffer of CAN peripheral.

```
CAN_MSG_RX_FRAME_ATTRIBUTE msgFrameAttr =
         CAN_MSG_RX_DATA_FRAME;
CAN1_RxCallbackRegister(CAN_Callback_Handler_Fup, 0,
         CAN_MSG_ATTR_RX_FIFO0);
```

The timestamp is then taken in the callback function after RM reception.

```
void CAN_Callback_Handler_Fup(uintptr_t context)
{
    if (CAN1_MessageReceive(&rx_messageID,
        &rx_messageLength, (uint8_t*)rx_message, 0,
        CAN_MSG_ATTR_RX_FIFO0, &msgFrameAttr) == true)
    {
        timestampValue = TC0_Timer32bitCounterGet();
    }
}
```

3.2 Offset Correction Methods

The literature provides two well-known software-based methods for offset correction on CAN that are implemented for CAN FD in this thesis. Gergeleit's method is described in Section 3.2.1 and the method based on the AUTOSAR standard is explained in Section 3.2.2. Implementation details are provided in Section 3.2.3.

3.2.1 Gergeleit's Method

The software-based offset correction method by Gergeleit [30] is illustrated in Fig. 3.2. Reference messages (RMs in gray boxes) with the message duration t_{RM} are sent by the TM node M with a period T and received synchronously by any CAN node (slaves and Master) at $t_k = k \cdot T$, $k = 0, 1, \ldots$. With each RM reception at t_k, M and each slave S_i take a timestamp of their respective local clock $t_{k,\text{M}}$ and $t_{k,\text{Si}}$. M then transmits $t_{k,\text{M}}$ with the next RM such that each slave S_i receives this timestamp at t_{k+1}. The difference between the received value $t_{k,\text{M}}$ and the stored value $t_{k,\text{Si}}$ is then used to perform the clock update

$$c_{\text{Si}}(t_{k+1}^{+}) = c_{\text{Si}}(t_{k+1}^{-}) + t_{k,\text{M}} - t_{k,\text{Si}}. \tag{3.1}$$

Here, $c_{\text{Si}}(t_{k+1}^{-})$ and $c_{\text{Si}}(t_{k+1}^{+})$ represent the local clock value of S_i before and after the update, respectively. For proper operation, RM must be transmitted in each cycle k. The advantage of Gergeleit's method compared to following method, AUTOSAR method, is that it only requires a single RM in each synchronization period T.

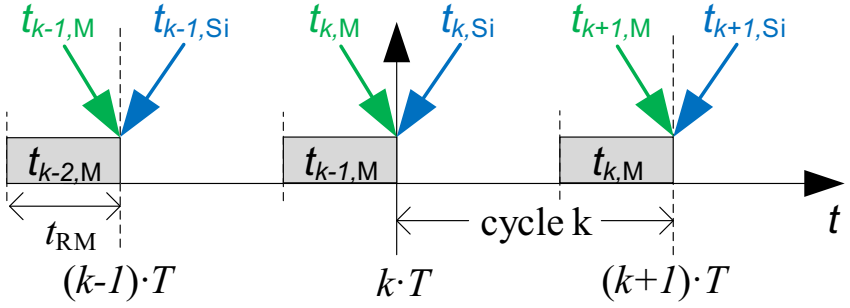


Figure 3.2: Illustration of RM transmission in Gergeleit's method.

In Fig. 3.3, the illustration of clock difference is shown if Gergeleit's method defined

in [30] is applied. To specify, this illustration shows the difference of the local clocks of TM and a slave node S_i. T represents the period of RM and d_{Si} represents the drift of the S_i local clock compared to local clock of TM. This drift causes a time difference of $d_{Si} \cdot T$ between two consecutive RMs. The boxes on the timeline in Fig. 3.3 shows the messages on the CAN bus. The blue boxes represent the RMs and the gray ones are the regular messages transmitted on the bus.

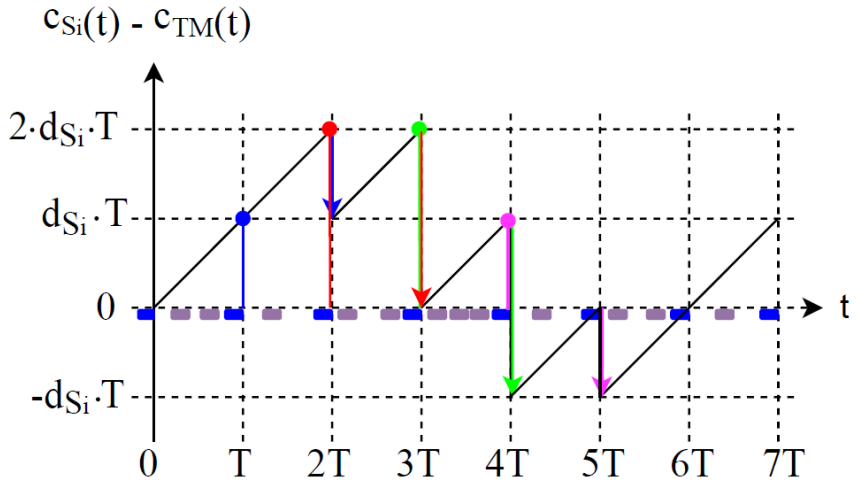


Figure 3.3: Illustration of clock difference in Gergeleit's method

The figure in 3.3 indicates there is no time difference between TM and S_i in the beginning. Therefore, at time T the slave makes zero correction. The first correction is applied at time $2 \cdot T$. The slave node corrects its local clock according to the timestamp taken at time T. The clock difference at time T is equal to $d_{Si} \cdot T$, that is the amount of correction. As it can be seen in the illustration, the theoretical upper limit guaranteed by Gergeleit's method [30] is $d_{Si} \cdot 2 \cdot T$ due to the approach of correction with respect to the previous timestamp. This approach also prevents this method from always fully compensating for the clock difference in every correction. This only happens if the previous timestamp has the same drift as the current RM.

Moreover, Gergeleit's method is used in several research works benefiting from its low bandwidth consumption and easy implementation in software. [54] applied Gergeleit's method with a 32-bit microcontroller, and it is claimed that a global clock is obtained with an approximate error of 4.5 microseconds . Nevertheless, having a global clock accuracy in the order of $4.5\,\mu$s is not possible without having very fre-

quent RMs which is not the desired case because of the higher bandwidth usage on CAN. The other possibility is having low clock drifts which does not agree with the real-world usage of the CAN bus. Additionally, there are fault-tolerant applications of Gergeleit's method in the literature [55]. The main steps of Gergeleit's method are summarized in Algorithm 1. Note that different functions are implemented for the TM and for the slaves.

Algorithm 1 Gergeleit's Method Implementation

```
1:  Function: master_main()
2:  Set Timer2 interrupt as RM period
3:  while 1 do
4:      if Timer2 reaches RM period then
5:          Send RM message that contains prevTime
6:          Save prevTime
7:      end if
8:  end while
9:  Function: slave_main()
10: Set CAN receive interrupt
11: while 1 do
12:     if RM received then
13:         Save masterTime from received message
14:         correctionTime = masterTime - prevTime
15:         correctTimer(correctionTime)
16:         Save prevTime
17:     end if
18: end while
```

3.2.2 AUTOSAR Method

The other offset correction method used in the thesis is the AUTOSAR method. According to the AUTOSAR specification [29], the master node sends a SYNC message right before the RM, also called as Follow-up (FUP), in each cycle k as shown in Fig. 3.4. Each slave node S_i takes a timestamp $\hat{t}_{k,\mathrm{Si}}$ right after receiving the SYNC

message and the master transmits its timestamp $\hat{t}_{k,\mathrm{M}}$ in the RM just upon the SYNC message. At the reception time $t_k = k \cdot T$ of the RM, each slave corrects its local clock based on the difference between timestamp of TM and its own timestamp:

$$c_{\mathrm{Si}}(t_k^+) = c_{\mathrm{Si}}(t_k^-) + \hat{t}_{k,\mathrm{M}} - \hat{t}_{k,\mathrm{Si}}. \tag{3.2}$$

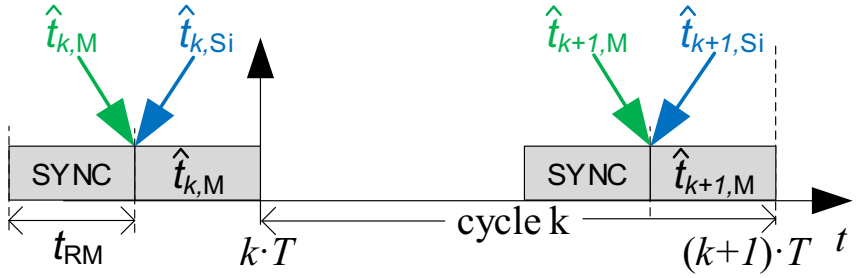


Figure 3.4: Illustration of the method according to AUTOSAR.

Different from Gergeleit's method, the AUTOSAR method requires two messages (SYNC and RM) per cycle k in order to take the timestamp close to the update time with the advantage of a more accurate offset correction.

The following illustration in Fig. 3.5, shows the clock difference when AUTOSAR clock synchronization method is applied. The orange boxes on the timeline represents the SYNC messages which are sent just before the following RM which are represented by blue boxes.

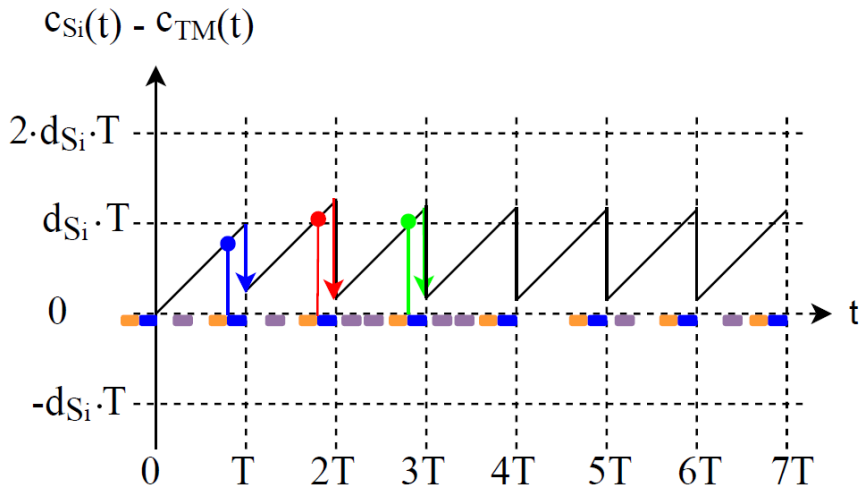


Figure 3.5: Illustration of clock difference in AUTOSAR method.

The main steps of the AUTOSAR method are summarized in Algorithm 2. Again,

there are different functions for the TM and for the slaves.

Algorithm 2 AUTOSAR Method Implementation

```
1:  Function: master_main()
2:  Set Timer2 interrupt as RM period
3:  while 1 do
4:      if Timer2 reaches RM period then
5:          Send SYNC message
6:          Save prevTime
7:      end if
8:      if SYNC message received then
9:          Send FUP message containing prevTime
10:     end if
11: end while
12: Function: slave_main()
13: Set CAN receive interrupt
14: while 1 do
15:     if SYNC received then
16:         Save myTime
17:     end if
18:     if FUP received then
19:         Save masterTime from received message
20:         correctionTime = masterTime - myTime
21:         correctTimer(correctionTime)
22:     end if
23: end while
```

There is a trade-off in AUTOSAR when compared to the Gergeleit's method. The positive side is the local clock difference after the correction is approximately zero which is the desired case. The negative side of the trade-off is, AUTOSAR uses two consecutive RMs, SYNC and FUP messages, in one cycle, which means occupying double bandwidth. In the illustration in Fig. 3.5, the clock drift correction error is the time during the FUP messages sent which is shown in blue boxes in the timeline. In a real application, this duration is much shorter compared to the RM cycle period

of T. Hence, the clock drift appeared in sending the FUP message, that is the extra amount of time causing to error after correction, is negligible with respect to the clock difference error $d_{Si} \cdot T$ that occurred in one cycle.

Lastly, it has to be emphasized that both methods can have better performance in clock accuracy if the RMs are sent more frequently, which decreases the T value. However, the desired goal is to achieve higher performance by using low bandwidth usage on the already limited CAN bandwidth.

3.2.3 Offset Correction Implementation

The difference between the two described offset correction methods is in the generation and reception of timestamps sent in RMs. For Gergeleit's method, a timestamp is sent in an RM in every period of T determined by the timer interrupt.

```
void TC4_Callback_InterruptHandler(TC_TIMER_STATUS status,
    uintptr_t context)
{
    refMes = true;
    message[0] = prevTime; // Send the previous timestamp
    CAN1_MessageTransmit(messageID,
     messageLength,message,CAN_MODE_FD_WITH_BRS,
     CAN_MSG_ATTR_TX_FIFO_DATA_FRAME);
}
```

By definition, the timestamps on the TM should be generated after the TM receives its own RM message. However, this is not possible in our implementation due to the lack of a receive interrupt emitted by its own transmission on the microcontroller boards used in this thesis. Hence, the timestamps on the TM are generated after the transmission interrupt of the RM, which indicates that the RM transmission is completed. This procedure is shown in the following callback function:

```
void CAN_Callback_Handler(uintptr_t context){
    timeRead = TC0_Timer32bitCounterGet();
```

```
    if(refMes){
        prevTime = timeRead;
        refMes = false;
    }
}
```

In the AUTOSAR method, the timestamp is sent in a follow-up message. That is, first, the RM with empty content (SYNC) is sent upon the timer interrupt emitted in every period of T.

```
void TC4_Callback_InterruptHandler(TC_TIMER_STATUS status,
        uintptr_t context)
{
    refTimeValue = TC0_Timer32bitCounterGet();
    dummyMessage[0] = 0;
    CAN1_MessageTransmit(syncMessageID,messageLength,
     dummyMessage, CAN_MODE_FD_WITH_BRS,
     CAN_MSG_ATTR_TX_FIFO_DATA_FRAME);
    syncMsg = true;
}
```

Then, with the successful transmission of the RM, the follow-up message is sent with the correct timestamp.

```
void CAN_Callback_Handler(uintptr_t context)
{
    timestampValue = TC0_Timer32bitCounterGet();
    if(syncMsg == true){
        message[0] = timestampValue;
        CAN1_MessageTransmit(fupMessageID,messageLength,
         message, CAN_MODE_FD_WITH_BRS,
         CAN_MSG_ATTR_TX_FIFO_DATA_FRAME);
        syncMsg = false;
```

```
    }
    if((fupMsg == true)){
        fupMsg = false;
    }
}
```

On the slave, the clock correction is carried out with the timestamp received in the RM for Gergeleit's method as follows. Since there is a delay of 2 cycles in getting the current counter value and updating it, this value is added to the counter.

```
void correctTimer(uint32_t update){
    TC0_Timer32bitCounterSet(TC0_Timer32bitCounterGet()
        + update + 2);
}

void CAN_Callback_Handler(uintptr_t context)
{
    myTime = TC0_Timer32bitCounterGet();
    if (CAN1_MessageReceive(&rx_messageID, &rx_messageLength,
        (uint8_t*)rx_message, 0, CAN_MSG_ATTR_RX_FIFO0,
        &msgFrameAttr))
    {
        masterTime = (int)rx_message[0];
        myTimeCorrection = masterTime - myPrevTime;
        myPrevTime = myTime;
        correctTimer(myTimeCorrection);
    }
}
```

Specifically, the timestamp in the RM is compared with the previous timestamp. Then, the difference between timestamps is corrected in the slave node. Differently, for the AUTOSAR method, the current timestamp is taken instead of the previous one when receiving the RM.

```
void CAN_Callback_Handler_Sync(uintptr_t context)
{
    if (CAN1_MessageReceive(&dummyrx_messageID,
        &rx_messageLength, (uint8_t*)dummy_message,
        0, CAN_MSG_ATTR_RX_FIFO0, &msgFrameAttr))
    {
        myTimestamp = TC0_Timer32bitCounterGet();
    }
}
```

Then, this timestamp is used for the clock update when receiving the follow-up message. Hereby, an offset of 15 clock ticks is added to compensate for software delays observed during experiments.

```
void CAN_Callback_Handler_Fup(uintptr_t context)
{
    if (CAN1_MessageReceive(&rx_messageID, &rx_messageLength,
        (uint8_t*)rx_message, 0, CAN_MSG_ATTR_RX_FIFO1,
        &msgFrameAttr))
    {
        masterTimestamp = (int)rx_message[0] + 15; // Add offset
        myNewTimeCorrection = masterTimestamp - myTimestamp;
        correctTimer(myNewTimeCorrection);
    }
}
```

3.3 Drift Correction Based on Timestamps

3.3.1 Drift Correction Background

In order to improve the clock accuracy of offset correction methods, it is required to realize drift correction. That is, in addition to the periodic clock corrections after the

reception of RMs as given in (3.1) and (3.2), the value of any slave S_i's LC also needs to be corrected between RMs. To this end, we note that the local clock c_{Si} of slave S_i will deviate by a clock difference Δc_{Si} from the local clock c_{M} of the TM after each clock correction with period T. That is, assuming that both clock values are identical at some time t, we can write the following relation.

$$c_{\mathrm{Si}}(t+T) - c_{\mathrm{Si}}(t) = c_{\mathrm{Si}}(t+T) - c_{\mathrm{M}}(t) = c_{\mathrm{M}}(t+T) + \Delta_{\mathrm{Si}} - c_{\mathrm{M}}(t). \tag{3.3}$$

That is, the local clock of S_i drifts with a time value of Δ_{Si} within a time interval of $c_{\mathrm{M}}(t+T) - c_{\mathrm{M}}(t)$ such that the clock drift d_{Si} of S_i with respect to M can be computed as

$$d_{\mathrm{Si}} = \frac{\Delta c_{\mathrm{Si}}}{c_{\mathrm{M}}(t+T) - c_{\mathrm{M}}(t)} = \frac{c_{\mathrm{Si}}(t+T) - c_{\mathrm{Si}}(t)}{c_{\mathrm{M}}(t+T) - c_{\mathrm{M}}(t)}. \tag{3.4}$$

It is clear that d_{Si} can be computed by the slave S_i by using the current and previous timestamps of TM and S_i. One way of compensating this drift is to perform a periodic local clock update with a period $\tau << T$. Considering a drift d_{Si} with its corresponding accumulated clock difference c_{Si} within one RM period T and , it holds that $N_{\mathrm{Si}} = T/\tau$ clock updates with the value

$$u_{\mathrm{Si}} = \frac{\Delta c_{\mathrm{Si}}}{N_{\mathrm{Si}}} \tag{3.5}$$

need to be performed between RMs. In summary, the proposed drift correction procedure is as follows.

1. Compute d_{Si} and Δc_{Si} after receiving a RM at time t,

2. Compute u_{Si} after receiving a RM at time t,

3. Apply clock updates with the value u_{Si} and period τ.

More precisely, the drift correction procedure is summarized in Algorithm 3.

Algorithm 3 Drift Correction Implementation

```
1:  Function: slave_main()
2:  Set CAN receive interrupt
3:  Set Timer4 interrupt for Drift Correction
4:  while 1 do
5:    if RM received then
6:      Save masterTime from received message
7:      Calculate correctionTime with CS algorithm
8:      masterTimeDifference = masterTime - prevMasterTime
9:      myTimeDifference = myTime - prevMyTime
10:     if myTimeDifference - masterTimeDifference > totalDriftCorrected then
11:       driftCorrection = -1
12:       driftCorrectionPeriod = masterTimeDifference / (myTimeDifference -
          masterTimeDifference - totalDriftCorrected)
13:     else
14:       driftCorrection = 1
15:       driftCorrectionPeriod = masterTimeDifference / (masterTimeDifference
          + totalDriftCorrected - myTimeDifference)
16:     end if
17:     totalDriftCorrected = 0
18:     prevMasterTime = masterTime
19:     prevMyTime = myTime + correctionTime
20:     Set Timer4 period to driftCorrectionPeriod
21:     correctTimer(correctionTime)
22:   end if
23: end while
24: Function: Timer4Interrupt()
25:   correctTimer(driftCorrection)
26:   totalDriftCorrected += driftCorrection
```

3.3.2 Drift Correction Implementation

Drift correction is implemented together with the offset correction method according to AUTOSAR. Following the explanation in the previous section, the clock drift is computed when receiving the follow-up message. After calculating TM's and S_i's timestamp changes in one RM period, the drift direction is decided. If the slave has a higher time difference than the master, the slave applies negative drift corrections in order to equalize the local clocks. The total corrected drift is taken into account in every RM period in order to correctly calculate the next drift. It is accumulated in each drift correction and cleared in every offset correction. Besides the drift correction, offset correction algorithms remains the same.

```
void CAN_Callback_Handler_Fup(uintptr_t context)
{
    if (CAN1_MessageReceive(&rx_messageID, &rx_messageLength,
    (uint8_t*)rx_message, 0, CAN_MSG_ATTR_RX_FIFO1,
    &msgFrameAttr) == true)
    {
        masterTime = (int)rx_message[0] + 15; //Add offset
        myNewTimeCorrection = masterTime - myTime;
        masterTimeDifference = masterTime - masterPrevTime;
        myTimeDifference = myTime - myPrevTime;
        if(myTimeDifference - masterTimeDifference >
            totalDriftCorrected){
            driftCorrection = -1;
            driftCorrectionPeriod = masterTimeDifference /
            (myTimeDifference - masterTimeDifference -
            totalDriftCorrected);
        }
        else{
            driftCorrection = +1;
            driftCorrectionPeriod = masterTimeDifference /
            (masterTimeDifference - myTimeDifference +
```

```
            totalDriftCorrected);
        }
        totalDriftCorrected = 0;
        masterPrevTime = masterTime;
        myPrevTime = myTime + myNewTimeCorrection;
        if(masterTimeDifference == 0){
            myNewTimeCorrection = 0;
            myPrevTime=0;
            driftCorrectionPeriod=400000000;
        }
        TC2_Timer32bitCounterSet(0);
        TC2_Timer32bitPeriodSet(driftCorrectionPeriod);
        correctTimer(myNewTimeCorrection);
    }
}
```

Using the computed parameters, drift correction is applied with the timer interrupt (TC2).

```
void TC2_Callback_InterruptHandler(TC_TIMER_STATUS status,
    uintptr_t context)
{
    correctTimer(driftCorrection);
    totalDriftCorrected += driftCorrection;
}
```

3.4 Weak TDMA

The realization of clock synchronization establishes a common time base for all nodes on a CAN bus. On the one hand, this enables the execution of simultaneous actions by different CAN nodes. On the other hand, this also makes it possible to coordinate the message transmissions of CAN nodes in order to increase the determinism of the communication and hence decrease message response times. This section first

explains the concept of weak TDMA for CAN that was introduced in [28].

3.4.1 Weak TDMA Background

We assume that the nodes on a CAN bus have synchronized clocks with a maximum clock difference of θ between any node and the global clock of a TM node. We note that such clock synchronization can be established by the methods described in the previous sections. Then, the medium-access method in [28] introduces an offset o_i and a transmission window size w_i for each message $M_i \in \mathcal{M}$. Hereby, the a-th instance of message $M_i \in \mathcal{M}$ is denoted as $M_{i,a}$. Writing $y_{i,a} = o_i + a \cdot p_i$ for $a = 0, 1, \ldots$, we define the time intervals

$$\mathcal{I}_{i,a} = y_{i,a} + [0, w_i), \tag{3.6}$$

during which the message instances $M_{i,a}$ can be transmitted. Accordingly, a message schedule consists of the set of offsets $\mathcal{O} = \{o_1, \ldots, o_m\}$ and the set of window sizes $\mathcal{W} = \{w_1, \ldots, w_m\}$. We further use the hyper-period (HP)

$$H = \mathrm{lcm}(p_1, \ldots, p_m), \tag{3.7}$$

which characterizes the time after which the schedule repeats. The number of repetitions of M_i per HP is $r_i = H/p_i$.

Depending on decision of $\mathcal{O}$ and $\mathcal{W}$, windows of different messages may overlap. Hence, we use feasible schedules in the sense that

1. $\forall i \in \{1, \ldots, m\}$, it holds that $L_i^{\max} \leq w_i$,
2. $\forall i, j \in \{1, \ldots, m\}$ with $i \neq j$, $a \in \{0, \ldots, r_i - 1\}$ and $b \in \{0, \ldots, r_j - 1\}$, it holds that $\mathcal{I}_{i,a} \cap \mathcal{I}_{j,b} = \emptyset$

That is, a feasible schedule ensures that 1) each message fits in its assigned window and 2) the windows of any two different messages do not overlap. That is, each message obtains exclusive access to the CAN bus during its assigned window, ensuring deterministic message transmission. In order to explain this fact, we introduce some additional notation, which is given for the duration of one HP. Specifically, the following time instants are needed:

- $y_{i,a}$: starting time of the window for $M_{i,a}$,
- $g_{i,a}$: time instant when the generation of $M_{i,a}$ is triggered by its node $\mu(M_i)$,
- $e_{i,a}$: time instant when $M_{i,a}$ is ready to enter arbitration on the CAN bus,
- $u_{i,a}$: time instant when the CAN bus is guaranteed to be free for $M_{i,a}$,
- $s_{i,a}$: time instant when $M_{i,a}$ starts transmission,
- $f_{i,a}$: time instant when $M_{i,a}$ finishes transmission including Inter Frame Spacing (IFS),
- $L_{i,a}$: length of the a-th instance of message M_i.

In addition, we recall the bound $\Delta_{\mu(M_i)}$ for the (MCU-dependent) maximum software delay between the time instant when the generation of message instance $M_{i,a}$ is triggered and the time instant when $M_{i,a}$ is ready to enter arbitration on the CAN bus. Specifically, it consists of the software tasks such as detecting the trigger, preparation of the CAN message content and its placement into the CAN controller transmission buffer. Furthermore, θ is the bound for the clock difference of any node relative to the global clock on the CAN bus.

Fig. 3.6 provides an illustration of the WTDMA operation and the notation introduced above. To this end, the figure shows the transmission of different instances of the consecutive messages M_i, M_j and M_k in two different HPs (denoted as HP_n and HP_{n+1}). The window start times are shown by dashed lines and the range of times where each message instance can enter arbitration is displayed by red and green arrows.

We note that this range is for example given by the interval $[y_{i,a} - \theta, y_{i,a} + \theta + \Delta_{\mu(M_i)}]$ for the message instance $M_{i,a}$. To point out the possible cases, $M_{i,a}$ starts transmission at the latest possible time $s_{i,a} = y_{i,a} + \theta + \Delta_{\mu(M_i)}$ in HP_n of the example. Then, $M_{j,b}$ starts transmission when the bus becomes idle right after the transmission of $M_{i,a}$ at time $s_{j,b} = s_{i,a} + L_{i,a} = f_{i,a} = u_{j,b}$. Here, it is important to note that, even if $e_{j,b} < f_{i,a}$, that is, $M_{j,b}$ is ready before the transmission of $M_{i,a}$ is completed, $M_{j,b}$ is transmitted after $M_{i,a}$ as long as $e_{j,b} > s_{i,a}$. Intuitively, this amounts to the fact that the time instant $s_{i,a}$ when $M_{i,a}$ starts transmission is guaranteed to be before the

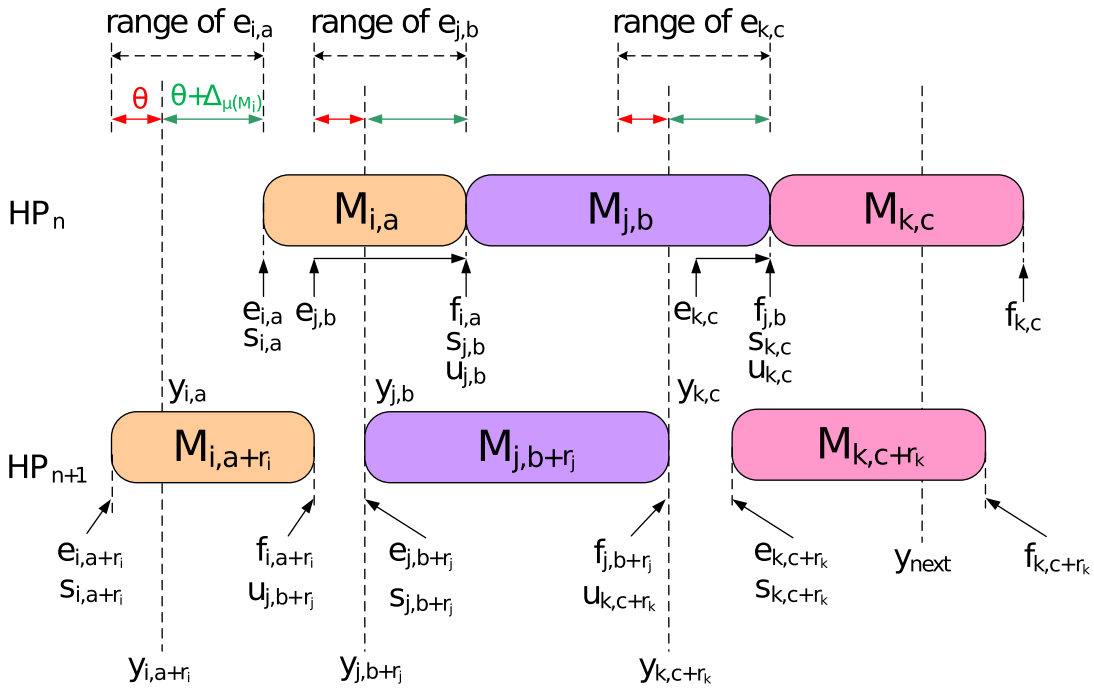


Figure 3.6: WTDMA notation and example illustration.

time instant $s_{j,b}$ when $M_{j,b}$ starts transmission. Similarly, $M_{k,c}$ starts transmission after the transmission of $M_{j,b}$ is completed at $f_{j,b}$. A different scenario is shown for HP_{n+1}. Here, the message instance $M_{i,a+r_i}$ is transmitted at the earliest possible time $y_{i,a+r_i} - \theta$. As a consequence, the CAN bus is idle at $u_{j,b+r_j} = f_{i,a+r_i}$ before $M_{j,b+r_j}$ is ready and $M_{j,b+r_j}$ is transmitted right at $s_{j,b+r_j} = e_{j,b+r_j}$. The same is true for $M_{k,c+r_k}$. At this point, we emphasize that a message can be ready during the time window of the previous message without affecting the WTDMA operation. That is, different from TDMA, the WTDMA operation does not require guard times between time windows for different messages and hence allows a more efficient bandwidth usage.

Using the above notation, we next quantify the correct operation of WTDMA. Consider that K messages are transmitted on the CAN bus according to a given WTDMA schedule. Then, we denote each message that is not transmitted in the specified order as a schedule violation and write

$$V = \frac{\text{Number of schedule violations}}{K} \tag{3.8}$$

for the ratio of schedule violations. We note that $V = 0$ is required for the correct

operation of WTDMA.

We further introduce the starting time delay

$$\phi_{j,b} = s_{j,b} - g_{j,b} \tag{3.9}$$

as the difference between the transmission start time $s_{j,b}$ and the trigger time $g_{j,b}$ of instance $M_{j,b}$.

Finally, the RT $R_{i,a}$ of instance $M_{i,a}$ is computed by adding the starting time delay $\phi_{i,a}$ and the actual message length $L_{i,a}$:

$$R_{i,a} = \phi_{i,a} + L_{i,a}. \tag{3.10}$$

Then, [28] shows that WTDMA operates correctly, that is, $V = 0$, if

$$w_{\min} > w_{\text{safe}} = \Delta_{\max} + 2 \cdot \theta. \tag{3.11}$$

Hereby, w_{safe} denotes the safe window size. Furthermore, the starting time delay $\phi_{j,b}$ and the RT $R_{j,b}$ of any message instance $M_{j,b}$ are bounded by

$$\phi_{j,b} \leq \phi^{\max} = \Delta_{\max} + 2\,\theta \text{ and } R_{j,b} \leq R_j^{\max} = \phi^{\max} + L_j^{\max}. \tag{3.12}$$

Specifically, this result provides a lower bound for the window size depending on the software delay $\Delta_{\max}$ and the maximum clock difference θ. Since the window size for each message $M_i \in \mathcal{M}$ depends on its payload size B_i, the smallest window size is obtained for messages with $B_i = 0$ as $55 \cdot \tau_{\text{bit}}$ (11 bit ID) and $80 \cdot \tau_{\text{bit}}$ (29 bit ID).

Considering realistic values of $\Delta_{\max} = 20\,\mu$s and $\theta = 5\,\mu$s for the software delay and the maximum clock difference, the minimum required safe window size $\Delta_{\max}+2\cdot\theta = 30\,\mu$s in (3.11) can be compared with the minimum message length w_{min} (11 bit ID) for different bit rates as shown in Fig. 3.7.

It is readily observed that the messages with minimum length are much larger than the minimum required window size even for large bit rates. That is, the proposed WTDMA method is expected to work well in practice.

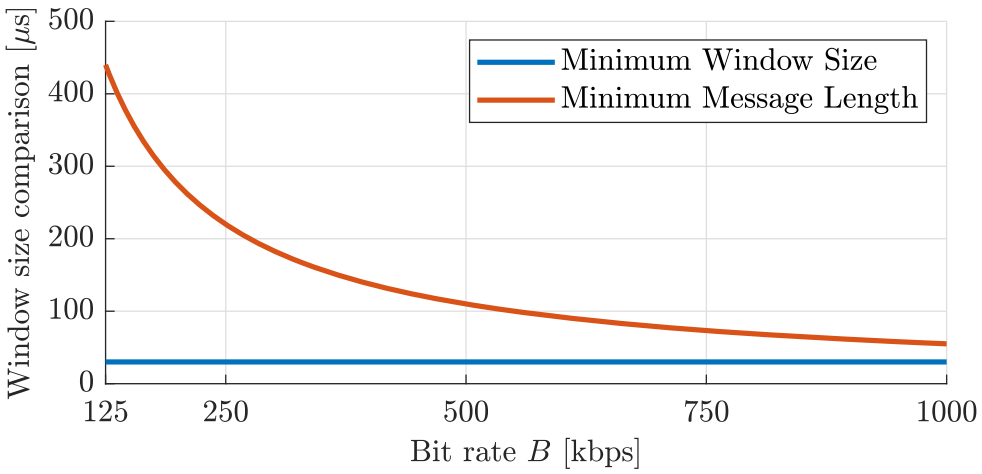


Figure 3.7: Minimum window size and minimum frame length.

Algorithm 4 Weak TDMA Implementation

```
1: Function:main()
2: while 1 do
3:     if Timer reaches any scheduled message then
4:         Send that scheduled message
5:         Update that scheduled message's time
6:     end if
7: end while
```

3.4.2 Weak TDMA Implementation

This part is constructed on clock synchronization methods as implemented in the previous section. After having an accurate global clock among the nodes, the preparation of a valid scheduler is required. The implementation of WTDMA for each message in the scheduler requires the realization of a periodic message transmission with a given offset and period. Determining the correct offset time is performed by polling the timer value for checking if the current time is passed by any of the offset of the message to be sent. After sending that message, offset value is updated by adding the period of that message.

```
while (true)
{
    timeControl = TC0_Timer32bitCounterGet();
    if(timeControl > offset0)
    {
        offset0 += period0;
        sendMessage(id0, size0, canfdEnabled0);
        if(offset0 >= 400000){ // Highest timer value
            offset0 -= 400000;
        }
    }
    else if(timeControl > offset1)
    {
        offset1 += period1;
        sendMessage(id1, size1, canfdEnabled1);
        if(offset1 >= 400000){
            offset1 -= 400000;
        }
    }
    .
    .
    .
}
```

This infinite while loop contains the number of messages that are transmitted by a node and hence need to be handled by the scheduler for that node. After any message's time has arrived, a message is sent to the bus with the corresponding message id, payload size, and message type which is a selection between CAN and CAN-FD.

```
void sendMessage(uint32_t canID, uint8_t messsageSize,
                 bool canFdEnabled)
{
    currentTime[0] = TC0_Timer32bitCounterGet();
```

```
    if(canFdEnabled)
    {
        CAN1_MessageTransmit(canID,messsageSize,currentTime,
        CAN_MODE_FD_WITH_BRS, CAN_MSG_ATTR_TX_FIFO_DATA_FRAME);
    }
    else
    {
        CAN1_MessageTransmit(canID,messsageSize,currentTime,
        CAN_MODE_NORMAL, CAN_MSG_ATTR_TX_FIFO_DATA_FRAME);
    }
    return;
}
```

CHAPTER 4

EVALUATION

This chapter evaluates the described implementation of clock synchronization and weak TDMA (WTDMA) for both CAN and CAN FD. First, Section 4.1.1 gives information about the test environment and Section 4.2 describes the experimental setup together with some properties of the used hardware such as oscillator drift. The performance of the clock synchronization is evaluated in Section 4.3 and the correct operation of WTDMA is verified in Section 4.4.

4.1 Development and Test Environment

In order to explain the experiments and the setup, brief explanation about the hardware and software tools used in the experiments is given in the following sections.

4.1.1 Microchip SAM E54 Xplained Pro Evaluation Board

For testing, identical evaluation boards, that are SAM E54 Xplained Pro series from Microchip [56] are used. These boards have a microcontroller from the SAM E54 family, ATSAME54P20A. This evaluation board includes an on-board embedded debugger and it needs no external tool for programming or debugging. The board has a USB serial converter that allows using serial communication directly from the USB port. It also has several external peripherals to extend the functionality supplied by the microcontroller. The SAM E54 Xplained Pro multifunctional evaluation board is shown in Fig. 4.1.

The board has ATA6561 CAN physical layer transceiver, produced by Microchip, that

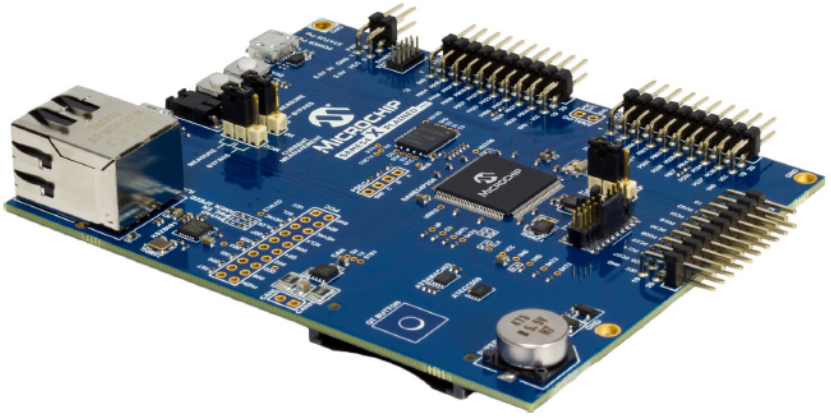


Figure 4.1: SAM E54 Xplained Pro evaluation board.

supports both CAN and CAN FD [56]. This property is the main reason of choosing this evaluation board for experiments. On-board CAN FD support makes this board favorable for this thesis work without requiring any external CAN FD controller.

There are two crystals on the board with frequency of 32.768 kHz and 12 MHz from the same supplier, Kyocera Crystal Device Corporation [56]. The first one is used for real-time operations and it is not the concern of our work. The 12 MHz crystal, with product number CX3225CA12000D0KPSC1, is used as clock source for the controller. There is also an internal clock generator inside the microcontroller, but it is not as stable as the external crystal.

In our experiments, three different peripherals of the controller are used. The External Interrupt Controller (EIC) peripheral is used as general-purpose input-output (GPIO) functionality for external triggering. It has to be noted that this functionality is not used as part of the implementation of clock synchronization and WTDMA, but it is needed for taking timestamps by the synchronous triggering from all the nodes. The Timer peripheral is used for realizing the local clock of each node. The capability of reading and updating the timer value at any time is crucial for the experiments. The nominal frequency of timers used in the experiments is 100 MHz, which equals to a period of 100 ns between each timer tick. The timers are used as a 32-bit timer by the concatenation of two 16-bit timers. Together, they can count up to approximately 430 seconds, which is slightly more than 7 minutes. For experiments that are longer than one timer cycle, the timers start from 0 after overflowing. The last peripheral used in

the experiments is the most essential one, the CAN peripheral. The board is capable of reading CAN and CAN-FD messages with the same command. For transmission, only a single parameter allows choosing the protocol between CAN and CAN-FD. The bitrate, sample point, and total time quanta are configurable. In the experiments, they are configured with exactly the same parameters for clean communication.

4.1.2 MPLAB x IDE Software Development Environment

MPLAB x IDE is developed by Microchip as the development environment for its own microcontrollers. It is an expandable, highly configurable software program that incorporates powerful tools to configure, develop, debug and qualify embedded designs for Microchip microcontrollers. This IDE has own configuration panel for easy application development. This configuration panel is used for changing parameters for peripherals supported by the microcontroller, that can be seen in Fig 4.2. MPLAB x IDE has the capability of importing and building multiple projects into a single workspace, which makes it easy to work with different projects. All the projects are developed in C programming language and built in XC-32 compiler of Microchip.

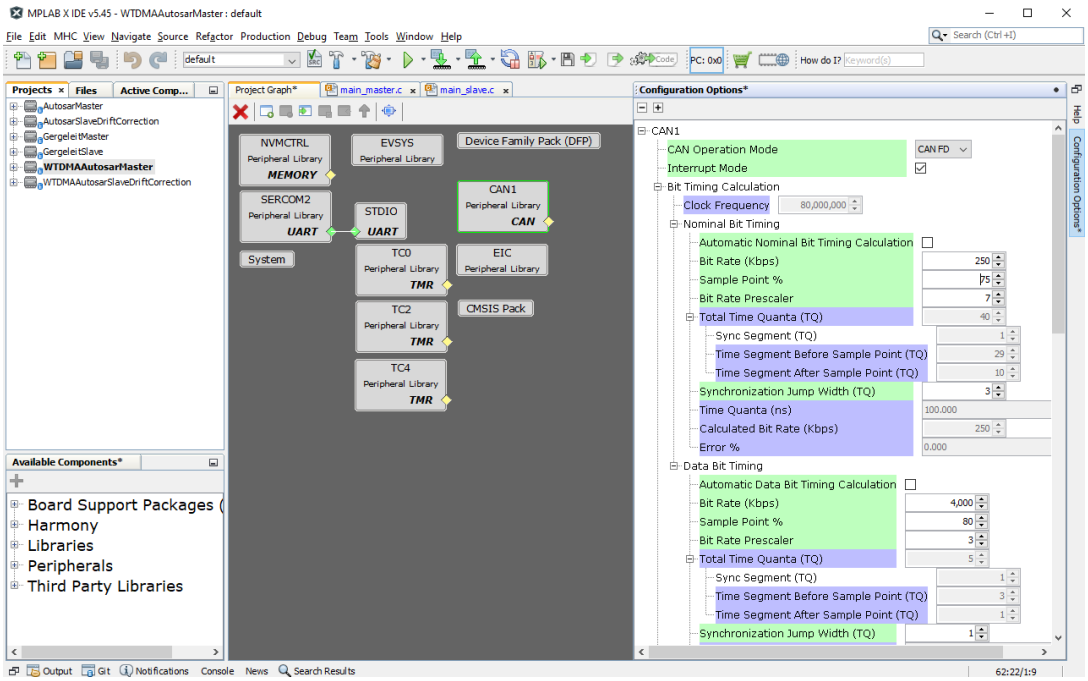


Figure 4.2: MPLAB x IDE Project Configuration interface

4.1.3 CAN Analyzer

PCAN-USB FD device is used for monitoring the CAN bus activities. It is capable of receiving and sending messages as well as analyzing the bus load. It can operate with both CAN and CAN-FD simultaneously after configuring the nominal and data bitrate values.

PCAN-View is the software developed by the same company, PEAK. In our projects, it is used for observing and recording the messages on the bus. It has several useful features, such as displaying the data in hexadecimal or decimal format, showing the maximum, minimum and mean values of bus load in a recording. It can save the records for further analyzing in external software. These records contain save time, CAN ID, message type, message length and payload data. These are more than enough parameters for analyzing the CAN operation.

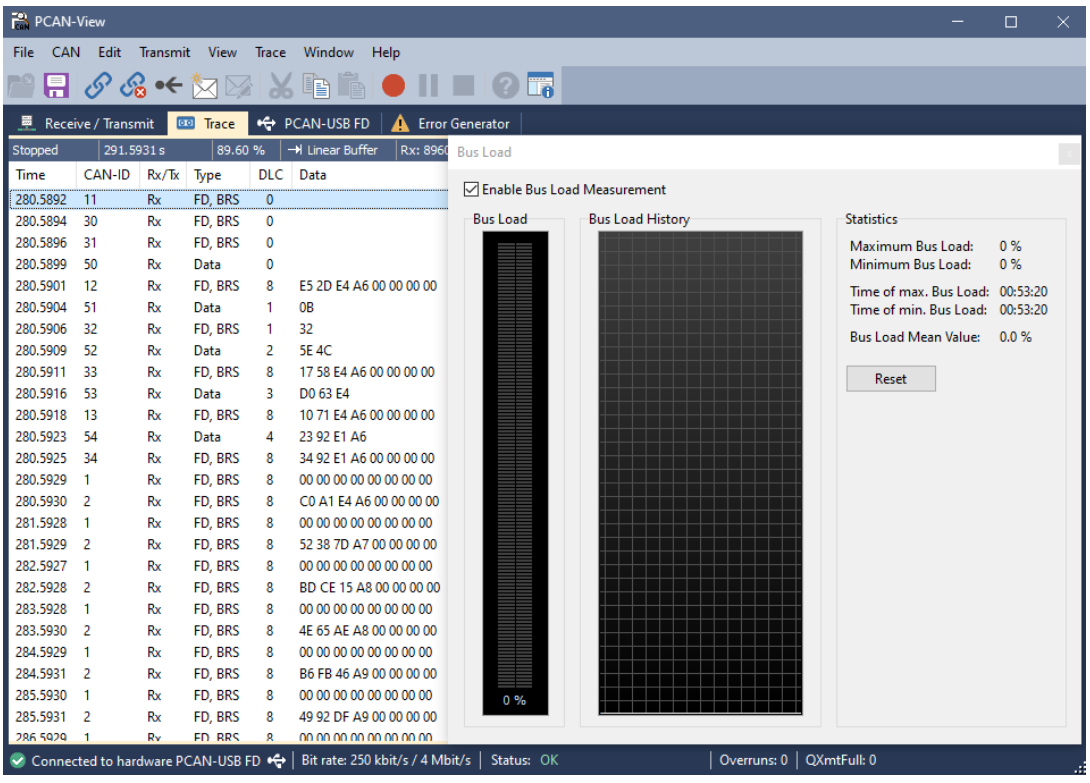


Figure 4.3: PCAN-View analyzer interface

4.2 Experiment Setup

There are 3 MCU boards [56] in the experimental setup. These boards are acting as CAN nodes TM, S1 and S2. They are connected to each other by CAN lines to communicate with each other. There is a CAN analyzer device, capable of monitoring CAN and CAN-FD messages in the bus. In real-world applications, the nodes are only connected by CAN bus. In our controlled experimental setup, there is also an external GPIO connection. This external connection ensures simultaneous timestamping. This is crucial for analyzing the performance of clock synchronization.

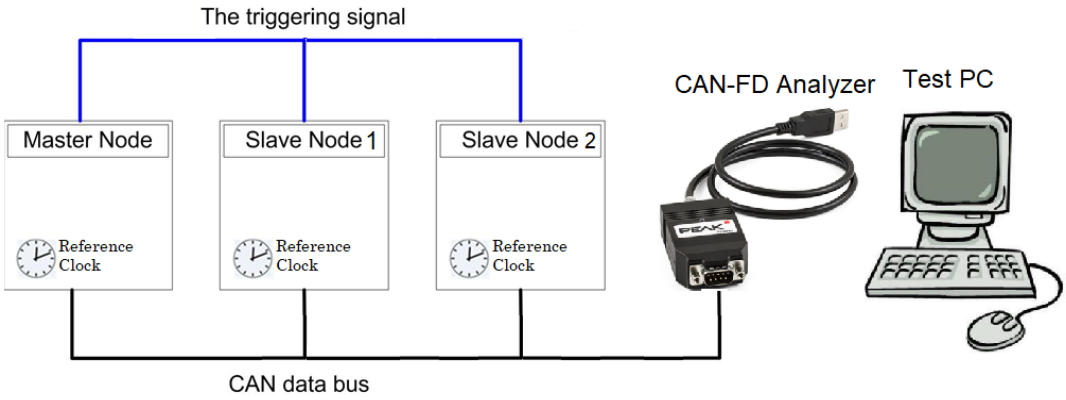


Figure 4.4: Experiment setup illustration

The motivation of this thesis work is the inevitable clock drift of local clocks of different microcontrollers. Accordingly, a clock drift measurement serves as the initial experiment conducted during the thesis work. In order to achieve clock synchronization, the nodes should correct their timer. The performance of this operation is measured by how much of the drift is corrected. To be more precise, the basis of the clock drift should be measured.

Evaluation boards have external crystals for feeding the clock sources of the controllers. Even though the crystals are from the same producer and have the same part number, there is again some time drift between the clocks. Each microcontroller's timer feature is used to measure its internal clock. The timer configurations are set up identically for conducting a controlled experiment. The timers are set to 100 ns tick period. An external signal is used for measuring the timer value at the same instant

for all the nodes. In this experiment, one node is used as a signal generator. That node can be called the master node. All other nodes, slave nodes, measure their timer value when the trigger signal arrives. The master node is also measuring its own time when the signal is received. GPIO feature of the microcontrollers is used for generating and receiving the signal. The period of this external signal, that is sampling period, is 20 microseconds. The result of the clock drift measurement is shown in Figure 4.5.

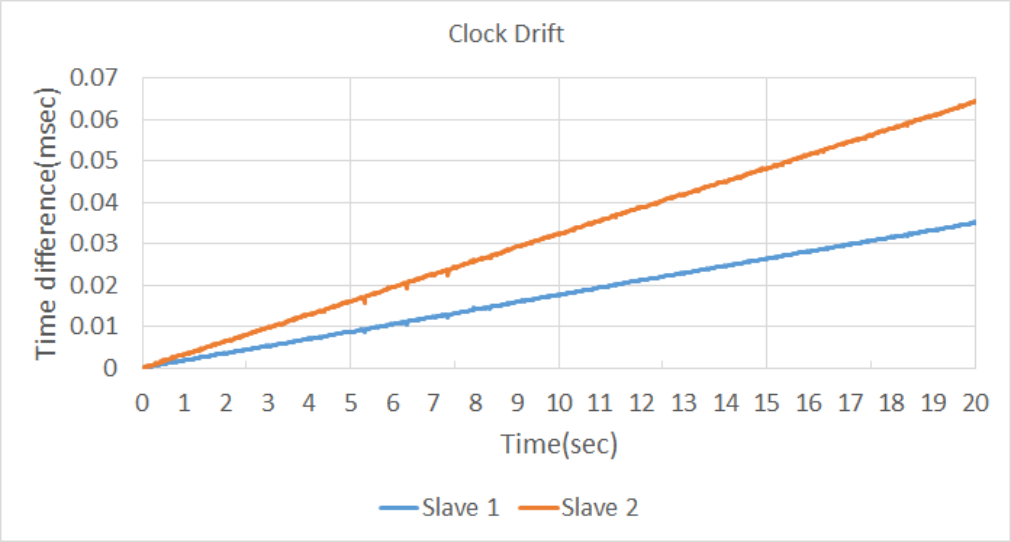


Figure 4.5: Slave nodes' clock drift with respect to master node

The result shows that the short-term frequency instability is very small as explained in Section 2.5. The measured clock drift has a linear shape. This means constant drift exists between the nodes. Since the boards are from the same vendor, the time drifts are quite low. For the first slave node, clock drift is ≈ 1.5 ppm. For the second slave node, this value is ≈ 3 ppm.

4.3 Clock Synchronization

Measuring the clock difference between the nodes shows that, there is a constant drift in local times which accumulates as time passes. In order to apply TDMA behaviour to the CAN bus, all messages on the bus should be transmitted within their allocated time. This deterministic bus access requires a synchronization mechanism. The essential element for this mechanism is a global clock with defined bounds on the time

difference between the nodes.

In this section, three different clock synchronization methods are experimented and analyzed in detail. The first method is Gergeleit's method [30], which is the first method proposed in the literature. The AUTOSAR method [29] is used in the second experiment. It has to be noted that the first two methods are only used for offset correction. That is, corrections are only applied once in a RM period, whereas the clock difference linearly increases between corrections. In order to overcome this problem, the last method is introduced. It is a hybrid method, where the slaves are running the AUTOSAR CS method and at the same time estimate the clock drift between each RM. In this method, the offset correction is assisted by corrections applied with respect to the drift estimations for further improvement.

All experiments are conducted in a controlled experimental setup, in order to analyze their performance correctly. Experiments are conducted by changing different parameters to understand the dependency on that parameter. Specifically, the RM period, CAN bit-rate and protocol (using whether CAN or CAN FD) are changed between the experiments.

4.3.1 Gergeleit's Method

The starting point of clock synchronization experiments is Gergeleit's method. In this experiment, there is one master and one slave node. According to Gergeleit's method, the master node is sending its local time as a reference time and slave nodes apply clock correction according to the difference between the reference time and their own local times. In this method, clock correction is applied with respect to the previous timestamp. This causes one RM period delay to the clock correction.

One of the key points to be emphasized in Gergeleit's method is, since the correction is applied depending on the previous timestamp, initialization time of timers should be equivalent. In the case of different starting times, Gergeleit's method is not capable of eliminating that time difference. In our experiment, an external GPIO signal is used for starting each node's timer at the same time for achieving the intended behavior from this method.

Timestamping has a crucial effect on the performance of Gergeleit's method. The timestamp should be saved at the start of receiving the reference message for both the master and slave nodes. In the experiment, the slave node is using an interrupt handler for receiving messages and it allows timestamping in the correct spot. However, for the master node, that is not possible to receive its own messages due to limitations caused by the evaluation board [56]. To achieve the optimum timestamping point, a transmission complete interrupt is used instead of the interrupt of receiving the reference message. This causes minor software errors in experiments as mentioned in Section 3.2.3. This software error occurred by the delay in receiving messages is not constant, therefore a perfect compensation is not possible at all times. In any case, the time difference is staying within some bounds.

When there are imperfections in the implementation, the illustration in Fig. 4.6 shows the expected effect of an error in timestamping e_{TS} [28]. The reason behind this behavior can be explained in this way. At $t = 0$, the time difference between master and slave node is zero. However, due to the errors, the slave node calculates the time difference as e_{TS}, and applies the wrong correction accordingly. Even if this inaccuracy in timestamping occurs only once, while other timestamps are recorded properly, the effect of this timestamping error at $t = 0$ remains in the following cycles. Gergeleit's method is sensitive to timestamping errors because it is operating depending on the previous errors. If some errors occur in operation, they accumulate such that the clock difference of master and slave regularly increase which is the opposite of the desired behavior of clock synchronization.

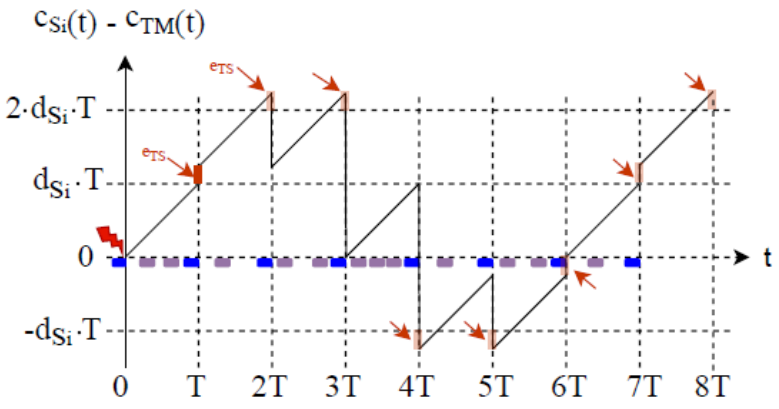


Figure 4.6: Illustration of the Gergeleit's method with a timestamping inaccuracy e_{TS}

4.3.2 AUTOSAR Method

The experiments are followed by another synchronization algorithm, that is called AUTOSAR method. This synchronization algorithm has advantages and disadvantages over Gergeleit's method. It has better results in terms of clock correction, however, it uses twice the bandwidth as Gergeleit's method. The main reason behind this fact is that AUTOSAR uses two RMs instead of one compared to Gergeleit's method. The timestamping importance is valid for AUTOSAR as well. Same experimental setup is used as in the previous experiment.

The other advantage over Gergeleit's method is, that the AUTOSAR method is fault tolerant in case of inequality of starting times. In the AUTOSAR method, correction is applied with respect to the time difference between two consecutive CAN messages whereas Gergeleit's method uses the timestamp taken at the previous RM. This allows that a node can join a running CAN setup any time and be synchronized with the master on the fly.

4.3.3 AUTOSAR Method with Drift Correction

Drift correction is conceptually different than the other clock synchronization methods. Both Gergeleit's and the AUTOSAR methods work as offset correction methods. They apply correction with respect to the RMs and update the local time once in every RM period. On the other hand, drift correction tries to eliminate the constant drift during the RM period. After estimating the drift value, correction is applied in the amplitude of a timer tick. This method is more sensitive to any case of error because having any error causes wrong drift estimation. Wrong drift estimation has a snowballing effect because the number of applied corrections is much more than offset correction methods.

4.3.4 Experimental Results

We first compare the clock difference measurements for the different methods during short measurement periods for eye inspection. Figure 4.7 shows the clock difference

measurements during a period of 20 s for both CAN and CAN FD. In these measurements, TM and S_1 are used. The bit rate for standard CAN is 250 kbps which is also equal to the speed of arbitration phase of CAN-FD. The data bit rate is 4 Mbps for CAN-FD. Each CS applied in measurements has 1 second for RM period.

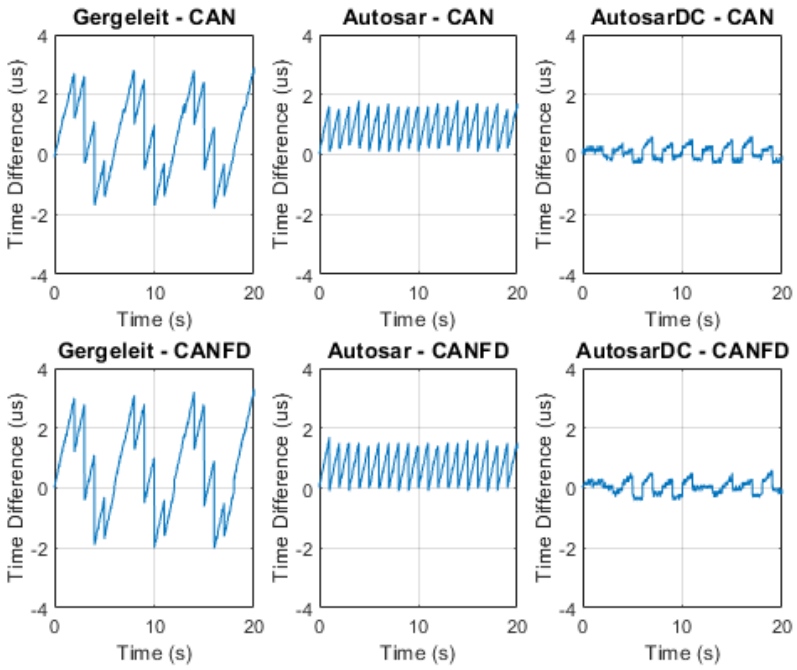


Figure 4.7: Comparison of clock synchronizations - Short Run.

Regarding Gergeleit's method, it can be seen that the clock differences evolve as expected from the conceptual plot in Figure 4.6. Here, it is interesting to observe that the results for CAN and CAN FD are very similar.

Looking at the results for the AUTOSAR method, it can be seen that the clock difference increases linearly between RMs and is reset to approximately zero with every RM. This is the case for both CAN and CAN FD.

When applying drift correction, the obtained clock differences depend on the quality of the drift estimate, which changes with the reception of each RM. Ideally, with a perfect drift estimation, the clock difference should stay constant after each offset correction according to the AUTOSAR method. Nevertheless, if the drift estimation is not perfect, a linear drift between RMs can still be observed. However, this drift is reduced compared to the drift without drift correction. Again, the results for CAN

and CAN FD are very similar.

In order to provide more insight on the long-term evolution of the clock differences, longer experiments in the order of 2000 s, which corresponds to more than 30 min were conducted. One representative measurement is shown in Figure 4.8.

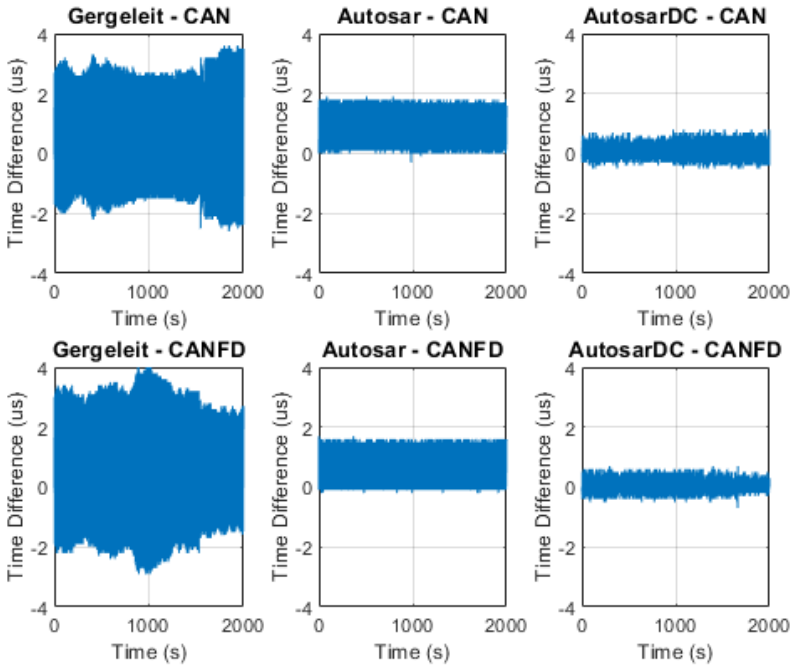


Figure 4.8: Comparison of clock synchronizations - Long Run

Looking at the results for Gergeleit's method, it is interesting to observe that the clock differences fluctuate over time. That is, here, the effect of inaccuracies as described above can directly be seen for both CAN and CAN FD. Differently, using the AUTOSAR method leads to predictable clock differences, which are only bounded by the almost constant oscillator drift. The maximum clock differences can be improved by applying drift correction, whereby the quality of the drift correction depends on measurement and timestamping inaccuracies.

This observation is further supported by the histogram plot in Figure 4.9. Here, it can be seen that the clock differences for Gergeleit's method are almost uniformly distributed between the upper bound of around $3.75\,\mu$s and the lower bound of about $-2\,\mu$s. This is expected since linear clock drift occurs between the RMs. The clock differences above and below the stated bounds have to be interpreted as outliers that

occur due to inaccuracies. A similar result can be seen for the AUTOSAR method but with smaller maximum values. When using drift correction, the clock differences are very tightly centered around $0\,\mu$s.

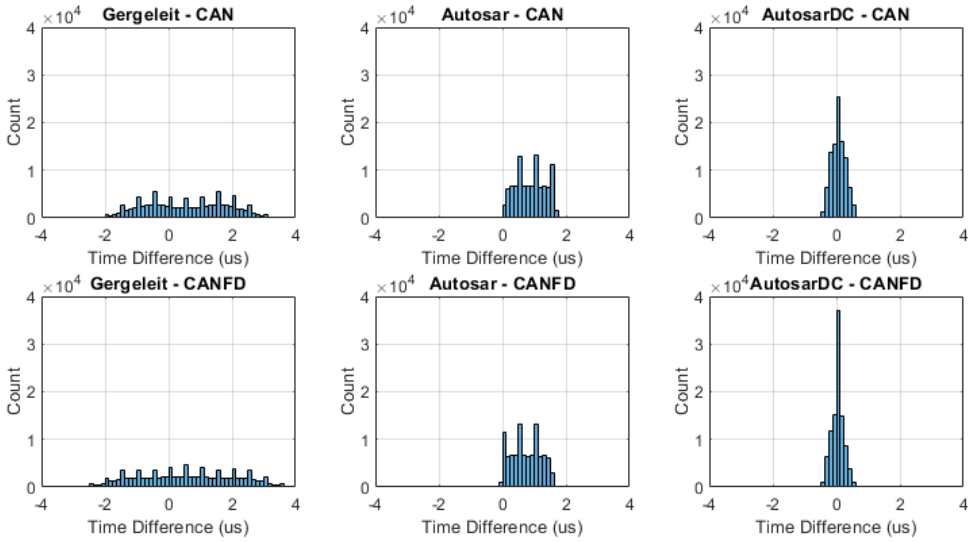


Figure 4.9: Comparison of clock synchronizations - Histogram

4.3.5 Statistical Evaluation

The illustrative measurements in the previous section were taken for a single CAN bus configuration. In this section, we conduct a detailed evaluation of the effect of different bus parameters on clock differences. Hereby, important performance metrics for each clock synchronization algorithm are the mean and maximum values of clock differences between master and slave nodes. The maximum time difference is the parameter taken into account when constructing a WTDMA schedule. Still, the mean value is a good indicator for evaluating the performances of the CS methods. We note that, all of the experiments are conducted for more than 30 minutes by recording 100000 samples from each node.

In Fig. 4.10, the effect of the RM period on the performance of the clock synchronization methods is shown. In the experiments, communication is performed by CAN FD with a nominal bit rate of 250 kbps and a data bit rate of 4 Mbps. The implementation imperfection leads to unexpected mean clock accuracy in Gergeleit's method.

However, the AUTOSAR and drift correction methods show the expected results. Specifically, for the AUTOSAR method, the maximum clock difference increases approximately with a factor of 2 when increasing the RM period, which is due to the longer duration of drifting clocks without offset correction. In addition, the mean error is more or less half of the maximum error, which is in line with the clock drift of the nodes. Here, it is beneficial to recall that the clock drift between TM and S1 node is approximately 1.5 ppm, which corresponds to an accumulated clock difference of 3 μs after a duration of 2 s. The drift correction method has a very low mean clock accuracy since it applies correction between RM periods. Most importantly, the clock accuracy of the drift correction method is almost not affected by changing the RM period. The reason for this result is that the measurement inaccuracies are reduced for larger RM periods, which leads to better drift estimates.

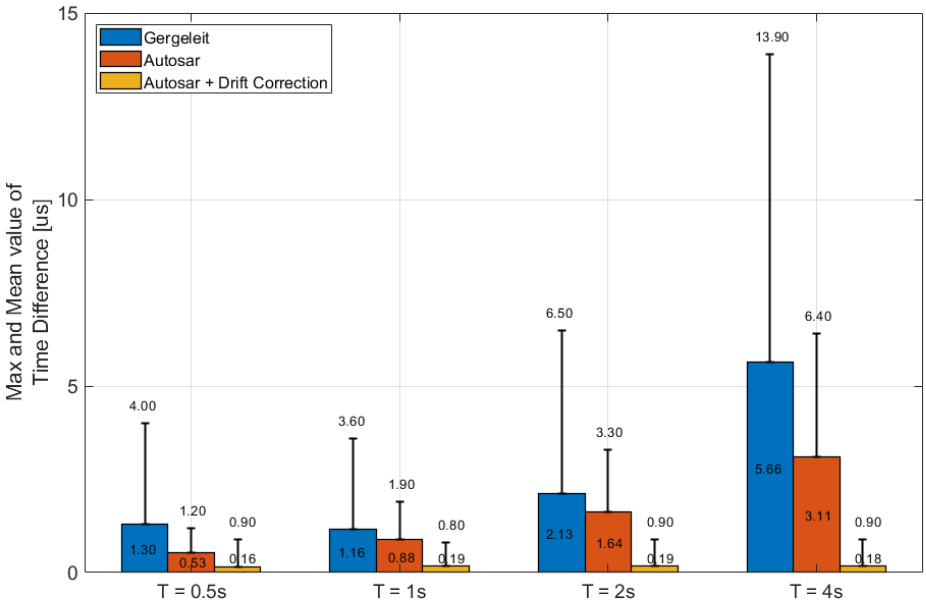


Figure 4.10: Clock synchronization performance of changing RM period

The other parameter to be explored is the bit rate of CAN messages. First, this parameter is analyzed for standard CAN protocol in three experiments. Afterward 4 of the commonly used CAN-FD bit rates are used. The related tests are performed with an RM period of 1 s. It is expected to have similar results while varying the CAN bitrate, which is confirmed in Fig. 4.11. The drift correction method has again the best result

when we analyzed the whole traffic excluding the first RM period. For comparing the mean values, the results are similar to the previous experiment.

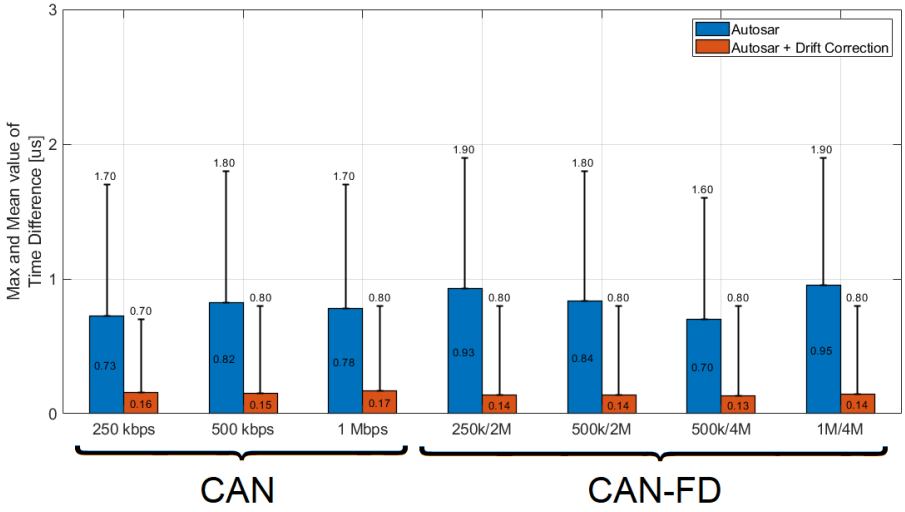


Figure 4.11: Clock synchronization performance of changing CAN bitrate

The last parameter is changing the communication protocol between CAN and CAN-FD. This experiment is important to show the possibility of using both protocols whenever needed. These tests are conducted with 250 kbps nominal bit timing and 4 Mbps data bit timing in CAN-FD counterpart. RM period is selected as 1 second. The results are similar in both maximum and mean time differences for all three methods. The key point is that, this experiment shows that the realization of clock synchronization is possible with CAN-FD without any significant performance degradation. This allows the implementation of WTDMA by using CAN-FD. which is superior to the standard CAN.

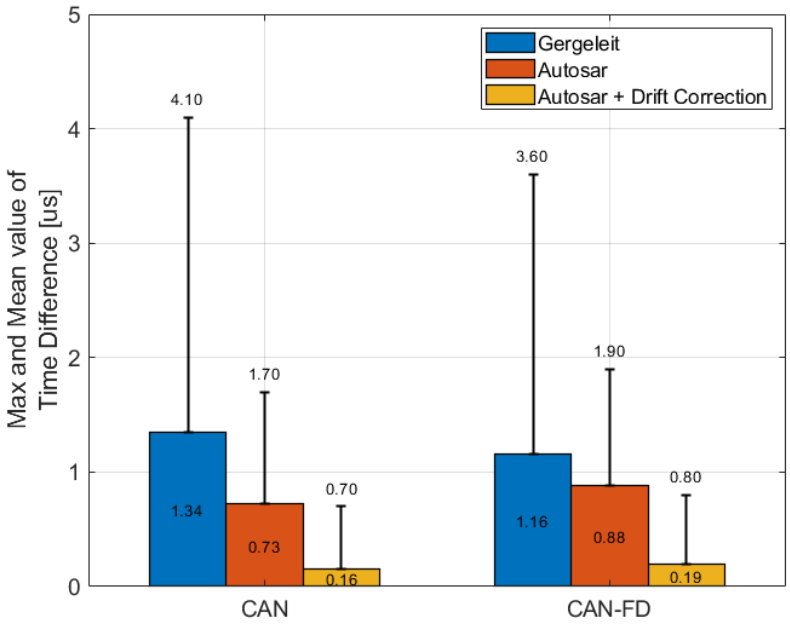


Figure 4.12: Clock synchronization performance of changing communication protocol

4.4 Weak TDMA

The realization of Weak TDMA based on successful clock synchronization is the ultimate goal of this thesis work. After achieving bounded clock differences, we can use the concept of a global clock between the nodes on the bus. The minimum window size for the scheduler can be calculated by the maximum message length, the current data bus speed and the maximum clock error using 3.11.

Experiments are performed with the setup defined in Section 4.2. The software implementation details are given in Sec. 3.4.2. The standard CAN bit rate and arbitration bit rate for CAN-FD is selected as 250 kbps. The data bit rate for CAN-FD is selected as 2 Mbps.

For analyzing the tests, the success criteria is the number schedule violations defined in 3.8. Also the theoretical maximum bus utilization as well as analyzed bus utilization values are given for better interpretation.

Table 4.1: Message and Schedule Properties

ID	o_i	p_i	l_i	ID	o_i	p_i	l_i	ID	o_i	p_i	l_i
10	1.08	5	0	30	1.52	5	0	50	1.96	5	0
11	1.30	5	0	31	1.74	5	0	51	2.44	5	1
12	2.18	5	1	32	2.70	5	1	52	2.96	5	2
13	3.90	5	3	33	3.26	5	2	53	3.56	5	3
14	15.00	20	8	34	4.62	5	4	54	4.24	5	4
15	15.54	20	8	35	5.00	20	8	55	5.54	20	8
16	40.00	200	8	36	10.00	20	8	56	10.54	20	8
17	80.00	200	8	37	20.00	40	8	57	20.54	40	8
18	120.00	200	8	38	40.54	200	8	58	80.54	200	8
19	200.00	1000	8	39	160.00	200	8	59	120.54	200	8
20	400.00	1000	8	40	200.54	1000	8	60	160.54	200	8
21	600.00	1000	8	41	400.54	1000	8	61	600.54	1000	8
1	0.00	1000	8	42	800.00	1000	8	62	800.54	1000	8
2	0.54	1000									

o_i and p_i are given in ms. $U_{\max}$ = 100% when $B = 250$ kbps

While realizing the WTDMA method, the message set in Table 4.1 is used. The message set includes messages with different payloads (l_i in bytes) size and periods (p_i in ms). The period of messages are changing between 5 ms and 1000 ms similar to existing message sets such as the SAE and PSA example message sets [15, 57, 58, 24]. The schedule has mostly 8 bytes of messages, on the other hand there are several messages with 0, 1, 2, 3, 4 bytes payloads. According to equation 3.11, short messages are more challenging for WTDMA. If the bit rate B equals to 250 kbps, in terms of bus utilization, the theoretical upper limit of this schedule is 100% due to bit stuffing. The minimum bus utilization is also calculated as 83.84% using the equation 2.2. The actual bus utilization has also recorded by the CAN analyzer for each test. Moreover, the offset of messages are arranged manually in order to obtain a feasible WTDMA.

Experiments are conducted such that 10^6 CAN messages are received by the CAN analyzer. Using this scheduler table, 3200 messages are sent in 1 second. This makes each experiment approximately 300 seconds long.

In the tests, AUTOSAR with Drift Correction CS algorithm is used. In Table 4.1, the messages in the first column are sent by the TM, where messages with ID 1 and 2 are the SYNC and FUP messages. The second and third column messages are sent by S_1 and S_2 respectively.

4.4.1 Evaluation of WTDMA for CAN

The first experiments are performed for showing the correct implementation of WTDMA. For this, two experiments are conducted. In order to run WTDMA successfully, the clock difference between nodes are bounded to some value. Using equation 3.11, for safe window size while using 250 kbps CAN bus, the upper limit of clock accuracy is calculated as 100 μs assuming the maximum software delay is 20 μs. Since all of the CS methods used in this thesis have far more better accuracy, no synchronization is used to analyze failure of WTDMA. In the first experiment, two slave nodes apply AUTOSAR with DC. In the second experiment one of the nodes, S_1, applies no correction at all. Table 4.2 shows the resulting schedule violations.

Table 4.2: Observed ratios of schedule violations.

	Exp1	Exp2
V	0	0.084

First experiment with ideal conditions has no schedule violations. This ensures that WTDMA concept is working. In the second experiment the schedule violations are coming from S_1 as it is not using any CS method. It is important to tell that, for the first 144 seconds there is no schedule violations. When looking at the clock accuracy at that time, a clock difference of 220 μs is monitored between S_1 and TM. After that time, only 0 payload messages are having schedule violations at first. From then, the ratio of violated messages starts to increase. The resultant schedule violation ratio is calculated around 300 seconds, which is the duration of the experiment. If the experiment went beyond, the violation ratio would increase since all the messages of S_1 cause violations. While the theoretical upper limit of the bus load is 100%, the analyzer recorded a mean of 87.2% utilization with a maximum of 88.7% and minimum of 85.5% during the tests. The reason for the deviations in the utilization is the bit stuffing, which depends on the message content and hence changes during the experiment.

In the third experiment, the message size of the following messages shown in Table 4.3 in the schedule is increased.

Table 4.3: Properties of Messages for Increased Bus Load

ID	B_i	ID	B_i	ID	B_i
10	1	30	1	50	1
11	1	31	1	51	2
12	2	32	2	52	3
13	4	33	3	53	4
		34	5	54	5

That is, there are no more messages with a payload of 0 Byte, and the calculated maximum bus load increased to 111.2% with this change. In the fourth experiment, a

different modification is applied to the schedule to show that bus utilization is not the only factor of affecting the message violation. The changes in Table 4.4 are applied in the fourth experiment. That is, the shortest messages in this schedule still have a payload of 0 Byte.

Table 4.4: Properties of Messages for Third Experiment

ID	B_i	ID	B_i
12	6	31	6

Looking at the results in Table 4.5, we can see that WTDMA can handle a bus load up to 97.5% in the third experiment. That is, increasing the payload of messages and giving them less window size than their maximum message length is compensated in WTDMA due to the varying bit stuffing as was also observed in [28]. Specifically, it is very unlikely that the maximum bit stuffing has to be applied for all messages at the same time, which is a known fact in the literature [59]. On the other hand, in the fourth experiment, schedule violations are observed at a maximum bus load of 97.0%. Even though the bus load is less compared to the third experiment, it has high violations such as 0.081 since the shortest messages of experiment 4 are shorter than the shortest messages of experiment 3. These violations happen due to the bus arbitration mechanism of CAN. If the delay of a message is longer than the total length of the message, the next message in the scheduler gains access to the bus if it has a higher priority ID.

Table 4.5: Observed ratios of schedule violations.

	Exp3	Exp4
V	0	0.081

4.4.2 Evaluation of WTDMA for CAN FD

CAN-FD has various advantages over standard CAN. Additional experiments are conducted to show this superior protocol, CAN-FD, is also capable of running WT-DMA schedule. In experiment 5, experiment 4 is repeated with all the nodes using

CAN-FD. Finally, in experiment 6, all the messages in Table 4.1 have 8 times more payload to show that CAN-FD can also run WTDMA at a maximum load just like standard CAN. The resulting schedule violations are shown in Table 4.6.

Table 4.6: Observed ratios of schedule violations.

	Exp5	Exp6
V	0	0

Here, it has to be noted that the bus load in experiment 5 is significantly decreased to a maximum of 70% since all messages are sent on CAN-FD. Moreover, no schedule violation has occurred. This result ensures that CAN-FD is capable of running WT-DMA scheduler. For the last experiment, it is the case that a maximum utilization of 87% is observed on the bus since all the messages contain an 8 times higher payload than the one specified in Table 4.1. Even though several messages have the maximum payload of 64 bytes, all the messages in the schedule are sent in their correct order during the test.

In all of these tests, one of the messages, for each node, in the schedule carries times-tamps for checking if the CS algorithm is working properly or not. The performance of CS algorithms is parallel to the results in the previous section. Moreover, the correct operation of the CS is confirmed by the fact that WTDMA works without problem.

CHAPTER 5

CONCLUSION

Controller Area Network (CAN) is still the most widely used in-vehicle communication bus in modern vehicles since it features advantages such as reliability, low cost, and simplicity. Considering that CAN is frequently used for safety-critical automotive applications which require high real-time Quality of Service (QoS), there are major shortcomings of CAN. First, CAN has a limited bandwidth, which is not sufficient for modern applications such as advanced driver assistance systems (ADAS) or autonomous vehicles. Second, CAN does not support clock synchronization and does not guarantee deterministic bus access.

The first shortcoming is mitigated by recent efforts to increase the bandwidth of CAN with the definition and hardware realization of new protocols such as CAN with Flexible Data-rate (CAN-FD) in 2012 and CAN with Extended Length (CAN-XL) in 2021. Specifically, CAN FD is already used in many vehicle applications. The second shortcoming is addressed in software by very recent work on highly accurate clock synchronization for CAN that further enables the realization of time-slotted access on CAN in the form of weak time division multiple access (WTDMA). Nevertheless, these developments are only performed for the standard CAN protocol but not for CAN FD.

This thesis implements and evaluates clock synchronization algorithms and WTDMA for CAN FD. The thesis first implements two different offset correction algorithms on recent microcontroller evaluation boards that support CAN FD. These methods are based on periodic timestamps which are expected to be taken at the same time by all nodes, and a time master whose local clock is assumed as the perfect clock broadcasts its timestamp by sending a reference message to slave nodes. The slave

nodes correct their local clocks periodically whenever they receive timestamps of the time master in reference messages. In addition, the timestamps can be used to perform drift correction to achieve a better clock accuracy. The evaluation of the implemented algorithms shows that a clock accuracy in the order of $1\ \mu$s can be achieved on CAN FD. These implementations show that CAN and CAN-FD have similar performances about clock accuracy. Next, the thesis uses the clock synchronization algorithms to realize WTDMA for deterministic bus access. In particular, WTDMA is implemented in software and hence it is fully compatible with the CAN FD protocol. The correct functionality of WTDMA is evaluated by hardware experiments. Most interestingly, these experiments show that bus loads above 97% can be achieved without violating the WTDMA schedule. In terms of maximum busload, both CAN and CAN-FD can reach up to limits. Moreover, CAN-FD is capable of transferring more meaningful data in similar bus utilization thanks to its capability of higher payload transmission.

Overall, the work in this thesis demonstrated that clock synchronization algorithms and a WTDMA scheme can be effectively implemented and evaluated on the CAN-FD protocol. The results showed that these techniques can achieve high levels of clock accuracy and support high bus loads, making them suitable for use in safety-critical automotive applications. The compatibility of these techniques with both CAN and CAN-FD adds to their versatility and potential for widespread adoption in the automotive industry. Moving forward, it would be interesting to explore the potential of these techniques for other applications beyond the automotive industry. The ability to achieve high levels of clock synchronization and deterministic bus access could be beneficial in other fields where real-time performance and reliability are important. Additionally, it would be useful to study the scalability of these techniques to large networks with many nodes, as well as their robustness in the presence of network failures or disruptions. Further research in these areas could help to extend the utility and impact of these techniques.

REFERENCES

[1] I. Standard-11898, "Road vehicles-interchange of digital information – Controller Area Network (CAN) for high-speed communication," *International Standards Organisation (ISO)*, 1993.

[2] Y. He, Z. Jia, M. Hu, C. Cui, Y. Cheng, and Y. Yang, "The hybrid similar neighborhood robust factorization machine model for can bus intrusion detection in the in-vehicle network," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–9, 2021.

[3] F. Amato, L. Coppolino, F. Mercaldo, F. Moscato, R. Nardone, and A. Santone, "Can-bus attack detection with deep learning," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–10, 2021.

[4] K. Agrawal, T. Alladi, A. Agrawal, V. Chamola, and A. Benslimane, "Novelads: A novel anomaly detection system for intra-vehicular networks," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–11, 2022.

[5] A. Michaels, V. S. S. Palukuru, M. J. Fletcher, C. Henshaw, S. Williams, T. Krauss, J. Lawlis, and J. Moore, "Can bus message authentication via co-channel rf watermark," *IEEE Transactions on Vehicular Technology*, pp. 1–1, 2022.

[6] F. Luckinger and T. Sauter, "Software-based AUTOSAR-compliant precision clock synchronization over CAN," *IEEE Transactions on Industrial Informatics*, vol. 18, pp. 7341–7350, Oct. 2022.

[7] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin, "Intra-vehicle networks: A review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 534–545, 2015.

[8] F. Hartwich, "CAN with flexible data rate," *Ihe international CAN Conference (iCC)*, 2012.

[9] R. B. GmbH, "CAN XL – the next step in CAN evolution." https://www.bosch-semiconductors.com/media/ip_modules/pdf_2/can_xl_1/canxl_intro_20210225.pdf, 2021. Accessed on 13.06.2022.

[10] M. Akpinar, K. W. Schmidt, and E. G. Schmidt, "Improved clock synchronization algorithms for the controller area network (CAN)," in *International Conference on Computer Communication and Networks*, pp. 1–8, IEEE, 2019.

[11] M. Akpınar, E. G. Schmidt, and K. Werner Schmidt, "Drift correction for the software-based clock synchronization on controller area network," in *2020 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–6, 2020.

[12] S. Mubeen, J. Maki-Turja, and M. Sjodin, "Worst-case response-time analysis for mixed messages with offsets in controller area network," in *Emerging Technologies Factory Automation, IEEE Conference on*, pp. 1–10, 2012.

[13] Y. Chen, R. Kurachi, H. Takada, , and G. Zeng, "Schedulability comparison for CAN message with offset: Priority queue versus FIFO queue," *RTNS*, pp. 181–192, 2011.

[14] H. Daigmorte and M. Boyer, "Evaluation of admissible can bus load with weak synchronization mechanism," in *Proceedings of the 25th International Conference on Real-Time Networks and Systems - RTNS '17*, (Grenoble, FR), pp. 277–286, 2017.

[15] K. Tindell and A. Burns, "Guaranteeing message latencies on controller area network (CAN)," in *In Proceedings of the 1st International CAN Conference*, pp. 1–2, CiA, 1994.

[16] R. Davis, A. Burns, R. Bril, and J. Lukkien, "Controller area network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Syst.*, vol. 35, pp. 239–272, April 2007.

[17] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Extending worst case response-time analysis for mixed messages in controller area network with priority and fifo queues," *IEEE Access*, vol. 2, pp. 365–380, 2014.

[18] R. Davis, S. Kollmann, V. Pollex, and F. Slomka, "Controller area network (can) schedulability analysis with fifo queues," in *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*, pp. 45–56, July 2011.

[19] H. Daigmorte, M. Boyer, and J. Migge, "Reducing CAN latencies by use of weak synchronization between stations," in *Proceedings of the 16th International CAN Conference*, pp. 1–8, 2017.

[20] K. W. Schmidt, "Robust priority assignments for extending existing controller area network applications," *IEEE Transactions on Industrial Informatics*, vol. 10, pp. 578–585, Feb 2014.

[21] M. Grenier, L. Havet, and N. Navet, "Pushing the limits of CAN - scheduling frames with offsets provides a major performance boost," in *European Congress on Embedded Real Time Software*, 2008.

[22] H. Daigmorte and M. Boyer, "Traversal time for weakly synchronized CAN bus," in *International Conference on Real-Time Networks and Systems*, pp. 35–44, 2016.

[23] G. Leen and D. Heffernan, "Ttcan: a new time-triggered controller area network," *Microprocessors and Microsystems*, vol. 26, no. 2, pp. 77 – 94, 2002.

[24] K. Schmidt and E. G. Schmidt, "Systematic message schedule construction for time-triggered CAN," *Vehicular Technology, IEEE Transactions on*, vol. 56, no. 6, pp. 3431–3441, 2007.

[25] N. Navet and F. Simonot-Lion, "In-vehicle communication networks: A historical perspective and review," in *Industrial Communication Technology Handbook*, pp. 50–1, CRC Press, 2017.

[26] M. Hadded, P. Muhlethaler, A. Laouiti, R. Zagrouba, and L. A. Saidane, "Tdma-based mac protocols for vehicular ad hoc networks: A survey, qualitative analysis, and open research issues," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2461–2492, 2015.

[27] O. Jubran and B. Westphal, "Optimizing guard time for tdma in a wireless sensor network - case study," in *39th Annual IEEE Conference on Local Computer Networks Workshops*, pp. 597–601, 2014.

[28] M. Akpınar, *A GENERAL FRAMEWORK FOR THE DETERMINISTIC MEDIUM ACCESS ON THE CONTROLLER AREA NETWORK*. PhD thesis, Middle East Technical University, 2022.

[29] AUTOSAR, "Specification of time synchronization over CAN – AUTOSAR CP release 4.3.1," Dec 2017.

[30] M. Gergeleit and H. Streich, "Implementing a distributed high-resolution real-time clock using the CAN-bus," in *International CAN-Conference*, 1994.

[31] F. Hartwich, "CAN with Flexible Data Rate," in *The international CAN Conference (iCC)*, 2012.

[32] "CAN with Flexible Data-Rate Specification version v1.0, Robert Bosch GmbH." `https://can-newsletter.org/assets/files/ttmedia/raw/e5740b7b5781b8960f55efcc2b93edf8.pdf`. Accessed on 2016.

[33] R. Lotoczky, "CAN-FD Flexible Data Rate CAN An Abbreviated primer." `https://vector.com/portal/medien/vector_cantech/Congress2013/1_9_CAN%20FD_Update.pdf`. Accessed on 2016.

[34] A. K. Sinha and S. Saurabh, "CAN FD: Performance reality," in *2017 3rd International Conference on Computational Intelligence Communication Technology (CICT)*, pp. 1–6, Feb 2017.

[35] G. M. Zago and E. P. de Freitas, "A Quantitative Performance Study on CAN and CAN FD Vehicular Networks," *IEEE Transactions on Industrial Electronics*, vol. 65, pp. 4413–4422, May 2018.

[36] "Renesas provides chips for Toyota." `https://can-newsletter.org/engineering/applications/171114_17-4_renesas-provides-chip-for-toytas-self-driving-cars_renesas`. Accessed on 2017.

[37] "Mercedes W140: First car with CAN." `https://can-newsletter.org/engineering/applications/160322_25th-anniversary-mercedes-w140-first-car-with-can`. Accessed on 2017.

[38] H. S. An and J. W. Jeon, "Analysis of CAN FD to CAN message routing method for CAN FD and CAN gateway," in *2017 17th International Conference on Control, Automation and Systems (ICCAS)*, pp. 528–533, Oct 2017.

[39] G. Cena, I. C. Bertolotti, T. Hu, and A. Valenzano, "Improving compatibility between CAN FD and legacy CAN devices," in *2015 IEEE 1st International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, pp. 419–426, Sept 2015.

[40] "CAN 2020: The future of CAN technology." `https://www.can-cia.org/news/cia-in-action/view/can-2020-the-future-of-can-technology/2016/3/21/`. Accessed on 2017.

[41] B. Cheon and J. W. Jeon, "The CAN FD network performance analysis using the CANoe," in *IEEE ISR 2013*, pp. 1–5, Oct 2013.

[42] M. Tenruh, P. Oikonomidis, P. Charchalakis, and E. Stipidis, "Modelling, simulation, and performance analyssi of a can fd system with sae benchmark based message set," *Ihe international CAN Conference (iCC)*, 2015.

[43] H. Kim, X. Ma, and B. R. Hamilton, "Tracking low-precision clocks with time-varying drifts using Kalman filtering," *IEEE/ACM Transactions on Networking*, vol. 20, pp. 257–270, Feb. 2012.

[44] K. S. Yildirim and A. Kantarci, "External gradient time synchronization in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 633–641, 2014.

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

[46] S. Park, H. Kim, H. Kim, C. N. Cho, and J. Choi, "Synchronization improvement of distributed clocks in EtherCAT networks," *IEEE Communications Letters*, vol. 21, pp. 1277–1280, June 2017.

[47] "Cardinal components inc. applications brief no. a.n. 419." http://www.cardinalxtal.com/static/frontend/files/cardinal-measuring-clock-stability.pdf.

[48] Z. Yang, L. He, L. Cai, and J. Pan, "Temperature-assisted clock synchronization and self-calibration for sensor networks," *IEEE Transactions on Wireless Communications*, vol. 13, pp. 3419–3429, June 2014.

[49] B. Martinez, X. Vilajosana, and D. Dujovne, "Accurate clock discipline for long-term synchronization intervals," *IEEE Sensors Journal*, vol. 17, pp. 2249–2258, April 2017.

[50] M. Akpınar, E. G. Schmidt, and K. Werner Schmidt, "Drift correction for the software-based clock synchronization on controller area network," in *IEEE Symposium on Computers and Communications*, pp. 1–6, 2020.

[51] S. Einspieler, N. Rathakrishnan, A. Prabhakara, B. Steinwender, and W. Elmenreich, "High accuracy software-based clock synchronization over can," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–9, 2021.

[52] M. Akpınar, E. G. Schmidt, and K. W. Schmidt, "Highly accurate clock synchronization with drift correction for the controller area network," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, pp. 4071–4082, Dec. 2022.

[53] "Cia 603 version 1.0.0 - CAN Network Time Management," 2017.

[54] B. Donnelly and J. Cosgrove, "Achieving microsecond accuracy with 32 bit microcontrollers using the controller area network (CAN)," in *IET Conference Proceedings*, pp. 508–513, January 2004.

[55] D. Lee and J. Allan, "Fault-tolerant clock synchronisation with microsecond-precision for CAN networked systems," in *International CAN Conference*, pp. 1–6, 2003.

[56] "Microchip SAM E54 Xplained Pro ATSAME54-XPRO." `https://www.microchip.com/content/dam/mchp/documents/OTH/ProductDocuments/UserGuides/70005321A.pdf`. Accessed on 10-09-2022.

[57] N. Navet, Y.-Q. Song, and F. Simonot, “Worst-case deadline failure probability in real-time applications distributed over controller area network,” *Journal of Systems Architecture*, vol. 46, pp. 607 – 617, 2000.

[58] J. Fonseca, F. Coutinho, and J. Barreiros, “Scheduling for a TTCAN network with a stochastic optimization algorithm,” in *International CAN in Automation Conference*, 2002.

[59] A. Batur, E. G. Schmidt, and K. W. Schmidt, “Evaluation of response time distributions for controller area network messages,” in *Signal Processing and Communications Applications Conference*, pp. 1–4, 2018.