NTNU
Norwegian University of
Science and Technology

FlexRay Analysis, Configuration Parameter Estimation, and Adversaries

Markus Iversen Huse

Master of Science in Cybernetics and Robotics
Submission date: June 2017
Supervisor: Sverre Hendseth, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Project Description

In this paper, the author presents an analysis of a time-triggered automotive communion network. The project was carried out in collaboration with Way AS, and the project vehicle used was a model year 2016 Mini One with a FlexRay communication network. The paper seeks to investigate the possibility of manipulating the original network communication in FlexRay. In order to accomplish this, the author shall:

- Develop the necessary tools in order to analyse the FlexRay communication network.
- Validate the information with the inherent error detection and format of a FlexRay frame.
- Estimate FlexRay configuration parameters.
- Investigate FlexRay message content.
- Research the possibility of performing a man-in-the-middle attack on FlexRay.

Trondheim, 04.06.2017

Markus Iversen Huse

Acknowledgments

I would like to especially thank Thor Henning Amdahl for providing technical assistance, academic discussions and inspiring perspectives.

Further, I would like to thank Way AS for permitting access to a test vehicle in order to analyse the on-board network.

In addition, I would like to express my sincere thank to SIMCO and Stein Håkon Lerdalen for supporting me with this paper by providing testing equipment, and technical assistance.

I would also like to thank Amelie Fritsch for academic support throughout the duration of this project.

Finally, I would like to thank Sverre Hendseth for supervising this paper and for allowing this research to be conducted.

Summary

This paper investigates the time-triggered automotive communication protocol FlexRay. Included is a study of on-board electronic vehicular systems, which focuses on possible vulnerabilities connected to networked data transmissions.

In order to provide information about the network communication and configuration parameters, a FlexRay analysis software was developed.

The analysis software is validated using the original FlexRay frame format from the protocol specification and the cyclic redundancy check (CRC) provided in the data transmission. Additionally, the results from the network analysis are compared to the communication of a FlexRay network where the configuration parameters and the frame content are known. The FlexRay analysis software was then used to investigate the FlexRay network of an actual vehicle.

Finally, based on the theory and experience from working with FlexRay, multiple adversary solutions are presented, as well as solutions for manipulating frame content of real-time communication in a FlexRay network.

Sammendrag

Denne oppgaven beskriver bilkommunikasjonsprotokollen FlexRay og elektroniske kjøretøysystemer med et fokus på mulige sårbarheter tilknyttet dataoverføring.

For å gi informasjon om datakommunikasjonen og nettverks-konfigurasjonsparametrene i FlexRay, ble et analyseprogram for FlexRay utviklet.

Analyseprogrammet valideres ved bruk av det originale FlexRay-datarammeformatet og den sykliske redundanskontrollen (CRC) i meldingsinnholdet i dataoverføringen. Analysen ble sammenlignet med kommunikasjonen i et nettverk der konfigurasjonsparametrene og meldingsinnholdet er kjent. FlexRay-analyseprogrammet ble videre brukt til å analysere nettverkskommunikasjonen i et fungerende kjøretøy.

Til slutt, basert på teorien og erfaringene opparbeidet under oppgaven, blir flere FlexRay-nettverksmotstandere presentert, i tillegg til løsninger for å manipulere meldingsinnholdet til FlexRay-meldinger i sanntid.

Table of Contents

Chapter 1

Introduction

1.1 Background and Motivation

Modern automobiles typically contain in excess of 60 Electronic Control Units (ECUs). These ECUs are dispersed throughout the vehicle and are connected to each other via multiple communication networks. The networks are used to distribute commands and information between the various vehicular systems. These electronic systems function in harsh environments, whilst at the same time providing different levels of security, tolerance, and demands for the data transmission [12].

Mechanical and hydraulic control systems are increasingly being superseded with electronic control. These new systems place a high demand on bandwidth, security and deterministic data transmission. Consequently, there is a real need for time-triggered architectures in automotive communication networks [12].

The automotive industry, in order to protect its research and inventions, operates almost exclusively with nondisclosure agreements (NDAs). Subsequently, the implementations of electrical vehicular systems are proprietary, i.e. neither the content nor the format of the data transmissions are known. However, encryption of network information is rarely implemented.

Nevertheless, a system vulnerability exists whereby an attacker could gain access to the bus system. The bus system transmits safety-critical data throughout the vehicle and this could endanger the vehicle occupants and the surrounding environment. Therefore, exposing these security vulnerabilities could raise an awareness with the automotive manufacturers and lead to

increased security in on-board electronic systems.

Time-triggered automotive communication networks, such as FlexRay, are prone to carrying safety-critical information as the main applications for FlexRay are related to X-by-wire systems. The FlexRay protocol uses a Time Division Multiple Access (TDMA) scheme for medium access, which ensures deterministic data communication throughout the vehicle network [12].

FlexRay is currently the most popular time-triggered automotive network protocol for X-by-wire applications. Its advantages includes high bandwidth, high tolerance and deterministic data communication. While the complexity of the execution flow in such networks (with a static transmission schedule) is reduced, the configuration complexity is greatly increased. Thus, analysing, or 'packet-sniffing', such a network protocol becomes increasingly difficult as opposed to an event-triggered architecture (e.g. CAN bus) [12].

This paper is written in collaboration with Way AS, who are developing a driving simulator, which utilises information from various electronic vehicular systems to supply information to the simulator. Therefore, an approach for retrieving information from the FlexRay network of the vehicle is desired. Additionally, an adversary solution for FlexRay is wanted in order to remotely control some of the vehicle's functions for the sake of simulating specific driving conditions. Moreover, implementing a FlexRay adversary in the vehicle would open the possibility of altering the original communication such that it would have data from the simulator. The driving simulator is shown in Figure 1.1.

This paper contributes to current research as analysis of FlexRay networks, without the use of proprietary systems, is quite sparse. Furthermore, a limited amount of research is published about the possibility of an adversary on FlexRay.

1.2 Goal and Method

The goal of this paper is to develop and implement the necessary software solution in order to analyse the network communication, frame content, and the FlexRay node configuration parameters of a FlexRay network. Furthermore, the goal is to show how an adversary could be implemented in a FlexRay network and how a specific frame or the entire network communication could be manipulated.

Figure 1.1: Driving Simulator[1]

In order to accomplish this, extensive research into the FlexRay protocol, automotive data security, and automotive electronic systems is required. Additionally, a particular focus is set on the frame transmission and network configuration in a FlexRay network as well as the physical layer of FlexRay.

1.3 Outline of the Report

Chapter 2 presents the literature and theory that are used in this paper. Whilst, Chapter 3 portrays the electronic systems and the main communication networks of the test vehicle. Chapter 4 describes the functionality of the FlexRay analysis software used to analyse FlexRay communication and configuration parameters. Furthermore, the analysis software from Chapter 4 is validated and used for the analysis of existing FlexRay networks in Chapter 5. The suggestions for FlexRay adversaries are then presented in Chapter 6. Finally, a discussion and the conclusion are given in Chapter 7 and Chapter 8, respectively.

It is not necessary to read sections 2.1 to 2.4 in order to understand the results presented in chapters 3 to 6, although it is recommended as it complements to the material presented in the rest of the paper.

[1] Picture Provided by Way AS

Chapter 2

Background

This chapter provides a general understanding of automotive communication networks, electronic systems, and automotive network security, with an emphasis on the specifications and applications of the FlexRay protocol. Examples and illustrations are used where they are suitable in order to give a better understanding of the subjects.

Prior to this paper, I have completed a broad literature study on the FlexRay protocol and developed a test network which implements two FlexRay nodes. I have also been working with data analysis of messages in Controller Area Network (CAN) and have a general understanding of automotive electronic systems.

This chapter is mostly based on the *"FlexRay protocol specification version 2.1"* [3] from the FlexRay Consortium, the pre-project of this paper *"On-Board Communication Systems in Vehicles - FlexRay"* [7], the book *"Vehicle Safety Communications - Protocols, Security, and Privacy"* [5] by Tao Zhang and Luca Delgrossi, as well as the book *"FlexRay and it´s Applications"* [12] by Dominique Paret.

Section 2.1, 2.2, 2.5.1 to 2.5.5, 2.5.9, 2.5.10, and 2.5.11 are also featured in the pre-project [7], however, are rewritten and complemented for the purpose of this paper.

2.1 On-Board Automotive Communication Networks

2.1.1 Controller Area Network

The Controller Area Network bus (CAN bus) is a low cost, high speed, high security communication protocol developed by Robert Bosch GmbH. The protocol was officially released in 1986 with the goal to solve current issues with on-board communication systems in the automotive industry. Preceding CAN bus, the vehicles were mainly running wire links between each electrical application and system controllers. Consequently, CAN bus led to a dramatic decrease in the amount of wiring needed in a vehicle and it introduced a network protocol that is able to handle the harsh environment of a vehicle. Today, CAN bus is still the primary choice regarding multiplexed communication networks for on-board communication [11].

Messages in a CAN bus cluster are broadcast to all nodes in the cluster. A message is defined by a message ID implicating the content of the message, not the origin. CAN bus starts a message transmission when a node initiates a transmission and has information to transfer. The message transmission is not confined to any timing scheme, and collision detection is achieved through the message arbitration inherent in CAN where a dominant bit state of a message will win the arbitration [5]. Consequently, the communication is structured around 'carrier-sense multiple access' in which all nodes in the cluster can initiate a message transmission, at any given point in time, and the message with the highest priority (lowest message ID) will be the dominant transmission [2]. Therefore, the CAN bus is an event based multi-master protocol. Transmission in CAN has a maximum bit rate of 1 MBit/s transferred over a twisted pair differential wire signal, which also achieves high immunity to electrical interference [5].

2.1.2 Local Interconnected Network

The primary purpose of the Local Interconnected Network (LIN) protocol is to function as a sub-network to the CAN bus at a lower cost, and with lower performance. LIN has a 'single master multiple slave' structure with data transfer rates of up to 20 KBit/s and message transmissions initiated by the master [11].

In a modern vehicle, the LIN might be used for minor functions as transmitting commands

from the driver window switch to the window engine while the driver door is connected to the CAN bus. This provides yet another security layer as a malfunction in the window engine will not affect the communication on the CAN-bus nor other communication networks as the malfunction would be stopped at the gateway. A malfunction in LIN has a smaller potential impact than a malfunction in a communication network responsible of transmitting engine or safety critical information. A modern vehicle typically contains around six LIN networks, which are communicating with different communication protocols through multiple gateways where the network performance and tolerance of CAN and FlexRay are not needed [11].

2.1.3 Media Oriented Systems Transport

The Media Oriented Systems Transport (MOST) was developed to replace older protocols for on-board audio links and was introduced around the same time as LIN. MOST is typically operating on the same gateways as LIN, CAN, and FlexRay, and provides a physical layer for the implementation of Ethernet in on-board communications. Through the gateway, MOST establishes communication between the radio module and safety, comfort and engine modules/ECUs [11].

2.2 X-By-Wire

The generic term 'X-by-Wire' is the general term for applications that are controlled by electrical wire links [12]. Inspired by fly-by-wire technology, X-by-wire are set to replace other systems, as e.g. hydraulic and mechanical, with systems that are controlled electronically. Further, for the automotive industry, the X-by-wire technology is applied to systems like steer-by-wire, brake-by-wire, suspension-by-wire, shift-by-wire and safe-by-wire. Consequently, the applications of X-by-wire usually contain highly critical systems which require a safety-critical data bus architecture with inherent fault tolerance at a high network speed [13].

X-by-wire is a technology used more frequently as the complexity and processor capacity of automotive systems are increasing and imposing higher demands on communication networks. Neither CAN, LIN nor MOST are suitable for X-by-wire applications mainly due to the lack of deterministic data transmission and network speed. For solving the current problems regarding deterministic data transmission and high speed data transmission in automotive systems, many standards have been considered such as CAN-FD, TTCAN, TTP/C, Byteflight and FlexRay. However, FlexRay is revealed as the preferred choice by most major automotive manufacturers [12].

2.3 Automotive Electronic Control Units

During the last couple of decades, mechanical and hydraulic systems in modern vehicles have been increasingly replaced by systems that are controlled electronically. The on-board systems of a vehicle are controlled by units called Electronic Control Units (ECUs), which are embedded systems that control one or multiple electronic systems. An ECU collects data from on-board sensors, performs calculations and distributes commands to other systems and ECUs in the vehicle in order to achieve appropriate and efficient driving performance and comfort in the vehicle. Communication between ECUs is established through automotive network protocols such as CAN bus, LIN, Ethernet (MOST) and FlexRay [5].

An ECU maintains almost every task of the vehicle´s functions, from turning on lights to more critical tasks such as autonomous brake systems. An ECU usually works independently, however, more complex tasks are in some cases divided over multiple ECUs [5].

An ECU is classified by its tasks. Hence, an ECU that controls the power train[1] is typically called 'power train ECU' and the ECU that controls the airbag deployment is called an 'Airbag Control Module' or an 'Airbag ECU'. This also accounts for the ECUs in combination with the comfort systems, infotainment systems and so forth [5].

2.4 Active Safety Systems

Passive safety systems, e.g. airbags and seat belts, are implemented in vehicles in order to minimise harm to the passengers in the event of a collision. However, today, passive safety systems seem to have reached their limit of effectiveness. Further, significant reduction in fatalities could be accomplished through electric systems and active safety systems [5].

An active safety system, as opposed to a passive safety system, takes action to prevent the initial collision or in order to minimise damage when a collision is unavoidable [5].

Anti-Lock Braking System The Anti-lock Braking System (ABS) adjusts the fluid pressure/flow to the brakes in order to maintain optimal braking performance and prevents the wheels from locking. This maintains traction with the road surface and helps the driver to steer the vehicle by preventing a skid and improving stopping distance. The ABS is located in an ECU of the vehicle, usually as its own module and the ECU regulates the brake fluid pressure depending on road conditions or imminent wheel lock. The ABS system often consists of wheel speed sensors at each one of the four wheels, hydraulic valves and an ECU [5].

Electronic Stability Control The Electronic Stability Control (ESC) detects loss of control over the vehicle and improves the vehicles stability by automatically controlling the brakes of each individual wheel in order to steer the vehicle. The ESC then improves the vehicles stability by employing the ABS in order to minimise skids [5].

Brake Assist Brake Assist Systems (BASs) are developed to assist drivers when applying the brakes in a panic situation. The excitation of the brake pedal is measured in order to detect a panic situation in which the driver is not applying enough force to the pedal. The BAS will then

[1] The components that generates and distribute power, such as engine, drive shafts and transmission [5].

boost the braking power and by that it reduces the stopping distance of the vehicle. BASs can also take action without input from the driver in situations were a collision is unavoidable. The system will then use a combination of warning signals in the form of light and sound before applying the brakes [5].

Further examples of active safety systems used in modern vehicles are Advanced Driver Assistance Systems, Adaptive Cruise Control, Blind Spot Assist, Attention Assist and various pre-crash systems. Those will however not be discussed in detail as they are not of great importance for the content of this paper and most of these systems are not present in the test vehicle.

2.5 FlexRay

2.5.1 Introduction to FlexRay

FlexRay is a time-triggered, high-speed, fault tolerant and deterministic communication protocol developed by the FlexRay consortium as a next-generation automotive communication network. The FlexRay protocol meets the demands for a communication network suitable to X-by-wire applications with faster and more reliable data transfers than CAN and LIN [5]. FlexRay is neither intended nor used as an alternative to CAN, LIN or MOST protocols, however it functions as a supplement to these networks where a higher bandwidth, more security or deterministic communication is needed [12].

2.5.2 The FlexRay Protocol

FlexRay is based on a multi-master structure for communication where bus access must comply with a strictly defined communication cycle and the participants of a FlexRay network are prohibited to have uncontrolled bus access. Each frame transmitted on the network is allocated in specific time slots with the use of Time Division Multiple Access (TDMA). The TDMA scheme of FlexRay determines dispatch times of all frames in a FlexRay cluster [16]. FlexRay can operate on a single or simultaneously on two communication channels where a preference for improved fault tolerance or additional bandwidth can be chosen [12]. The persistent timing of the TDMA scheme in FlexRay ensures deterministic transmission in which data arrives on the microsecond in a predicted time frame. Further, the FlexRay protocol handles a pre-defined communication cycle that includes pre-defined space for both static and dynamic segments. Consequently, the configuration of network parameters must be known to all nodes in a FlexRay cluster in order to communicate in the network [8].

The quality of the transmitted frames is determined by a Cyclic Redundancy Check (CRC), which then excludes the problems that occur with round trip signal propagation time [12]. FlexRay has an inherent ‘never give up’ strategy in which, if a problem occurs in the network, the system always tries to operate and therefore never stops trying [8].

2.5.3 The FlexRay Communication Cycle

The FlexRay data transmission is configured around a media access scheme called the communication cycle. The communication cycle is a fundamental element of the FlexRay communication in which each cycle is identical in length and includes a fixed duration of static and dynamic segments. The length of a communication cycle is generally around 1 to 5 ms and is configured by the network designer [16][8].

2.5.3.1 The Static Segment

The static segment occurs in the beginning of the communication cycle and is consisting of a fixed number of equal length static slots (maximum 1023). The length of a static slot is defined by the network designer. The static slots are assigned to the FlexRay messages that will be transmitted during the static segment. The slots are identified by a slot number and a message ID, whereas the FlexRay node has a slot counter. The slot counters on all nodes of a FlexRay cluster are incremented synchronously at the start of every static slot to ensure that all messages are transmitted at the correct time and in the proper sequence in each communication cycle. Consequently, the static segment is predestined for the transmission of real-time messages and is operating with Global TDMA (GTDMA) for the medium access [16] [12].

2.5.3.2 The Dynamic Segment

Following the static segment, a new segment occurs called the dynamic segment. The dynamic segment has a fixed duration in the communication cycle and is consisting of 'minislots' similar to the static slots of the static segment. Data transmission during the dynamic segment is not forced, however, transmitted when needed, e.g. triggered by an event. When a dynamic message is sent, the slot counter is incremented upon receiving that message. If a frame is not transmitted during a minislot, the slot counter corresponding to the minislot will be incremented after a defined time period [16]. Medium access during the dynamic segment is of type Flexible TDMA (FTDMA), which then enables the segment to secure a time slot for the message transmission. A prioritisation system is used for the medium access where a higher prioritisation of a frame (lower message ID) is sent first. Further, the prioritisation system ensures that no data collision

or bus arbitration will occur during the dynamic segment [12].

An illustration of the static segment and the dynamic segment with the corresponding static slot and minislot is shown in Figure 2.1.

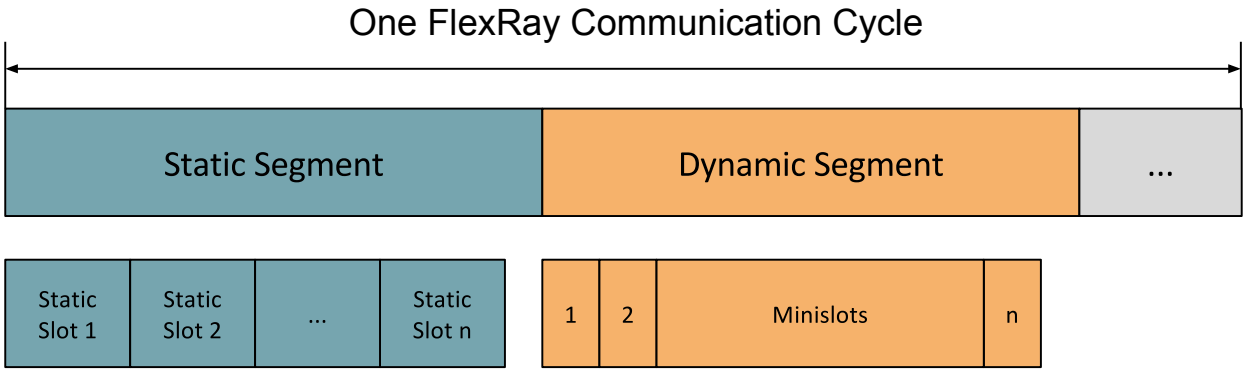

Figure 2.1: Static and Dynamic Segment

Two additional segments are added in addition to the static and dynamic segment to complete the communication cycle, the 'Network Idle Time' (NIT) segment and the optional 'Symbol Window' segment as illustrated in Figure 2.2. The symbol window overlooks the performance of the bus guardian and occurs right after the dynamic segment [16]. A transmission of a symbol is dedicated to verify the functioning of a local bus guardian with the use of a 30 bit Media access Test Symbol (MTS) followed by a channel idle delimiter (CID) (for CID see Section 2.5.8.2) [12]. The NIT segment keeps the transmission lines idle (no data transmission) and allows the FlexRay nodes to calculate the factors needed in order to synchronise the local clocks. Further, an offset error correction is produced at the completion of the NIT [16].

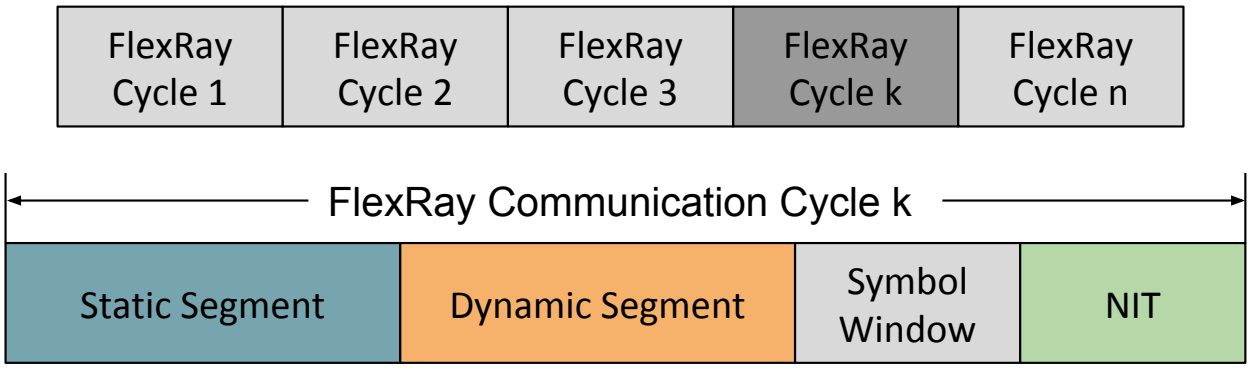

Figure 2.2: FlexRay Communication Cycle

2.5.4 Network Topology

The unconstrained topology specification of a FlexRay cluster allows different structures. A cluster could consist of a passive bus structure, active star topology or a combination of passive and active star called hybrid topology [3].

The passive bus topology is familiar as it is commonly used by both CAN and LIN, and therefore a FlexRay implementation requires little or no alteration of the network cabling in a vehicle. In a passive bus topology all nodes are connected by a single network cable with branches out to the ECUs. Further, the end nodes typically carry out the bus termination [8].

In a FlexRay cluster with active star topology, each ECU is connected to a central node (the active star) through individual network links. The maximum distance between nodes is then increased and a network could be constructed such that a node could malfunction without bringing down the network. The active star topology could also help reducing the amount of external electrical interference by introducing flexibility in the wiring network in order to circumvent areas with a high potential for electrical interference [8].

An illustration of a passive bus topology and active star topology is shown in Figure 2.3.

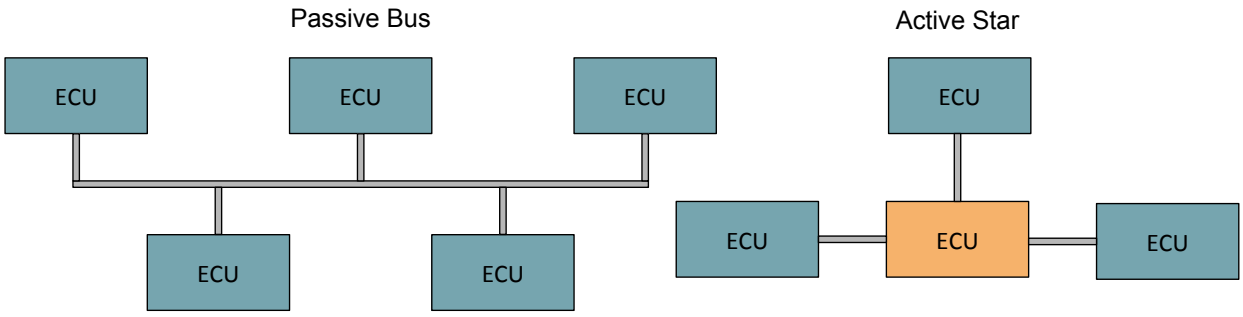

Figure 2.3: Passive Bus and Active Star Topology

2.5.5 Node Synchronisation in FlexRay

FlexRay follows the paradigms of time-triggered communication architectures where the underlying logic consists of system actions that are triggered at a specific point in time during the cycle. For a cluster-wide node synchronisation, a distributed fault-tolerant clock synchronisation mechanism is necessary [16]. The methods used in FlexRay in order to synchronise the local time

base of the nodes are based on offset correction and rate correction, executed the same way in all nodes [3].

2.5.5.1 Macrotick and Microtick

The smallest practical time unit in a FlexRay network is called a 'macrotick' (MT) and network participants synchronise and accommodate their local clocks in order to establish the macrotick simultaneously in all nodes in the network. Consequently, the macrotick is used for the global time synchronisation of a FlexRay network. The synchronisation of the macrotick has the advantage that the macrotick dependent data is also synchronised. The duration of the macrotick is specified by the network designer, typically around 1 μs long [8]. As a smaller FlexRay time unit, the 'microtick' (μT) is calculated in each node from the local clock and will only be influenced by drifts and tolerances of the local oscillators. Further, the μT is different on every node in a FlexRay cluster and performs as a local time unit. As the μT is the lowest time unit in FlexRay, the precision of a node correlates with the μT [12].

Figure 2.4 shows an illustration of the FlexRay macroticks within a communication cycle.

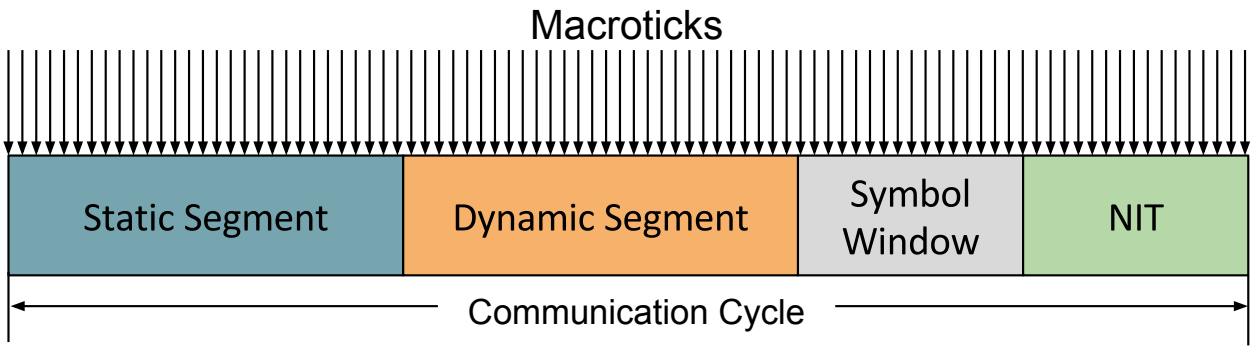

Figure 2.4: FlexRay Macroticks

2.5.5.2 Network Startup Synchronisation

Nodes in a FlexRay cluster are synchronised with the use of 'startup frames' and 'sync frames' eliminating the need for an external synchronisation clock signal. During the start phase of the network, two nodes are needed to transmit a startup frame in order to initiate the transmission of the remaining nodes. The action of transmitting startup frames is recognised as a 'coldstart'

identifying the nodes in control as coldstart nodes [8].

One of the coldstart nodes will dominate the start phase of the network with its local clock and will initiate the data transmission on the network in coldstart listen mode. When in coldstart listen, the node verifies that no data transmission is present on the network, while further sending a Collision Avoidance Symbol (CAS) in order to instruct the other nodes in the FlexRay cluster of the presence of a leading coldstart node. The first startup frames are sent for a minimum of four communication cycles on both channels by the leading coldstart node enabling the other coldstart nodes to send their sync frames in order to achieve cluster-wide network synchronisation and completing the startup phase [12]. After completion of the startup phase the remaining nodes will wait for the transmission of sync frames in order to calculate the time difference between the sync frames and subsequent frame transmission [8]. Each startup frame is required to also be a sync frame, hence, each coldstart node is also a synchronisation node [3].

2.5.5.3 Node Synchronisation

The offset and rate correction takes place when the startup phase is complete and is used to preserve and re-calibrate accurate cluster-wide synchronisation [8].

The offset correction is performed during the NIT segment and is synchronising the start time of the communication cycle. The synchronisation is done with adding/subtracting a multiple of microticks from the offset correction portion of the NIT segment as calculated by the clock synchronisation algorithm. The offset correction is calculated every communication cycle, however, only applied in the end of the odd communication cycle. The rate correction is adding or removing a microtick integer of the total number of microticks in a communication cycle configured as a function of the local clock of a node. The rate correction is calculated once over a double communication cycle duration (even-odd) and corrected during the NIT segment [3].

2.5.6 FlexRay Message Format

The FlexRay frame consists of three main segments; a header segment, payload segment, and trailer segment. A FlexRay node transmits the FlexRay frame starting with the the header segment, then the payload segment, and at last the trailer segment. All segments are transmitted from left to right, starting with the first leftmost bit of each segment as illustrated in Figure 2.5. The

FlexRay frame is used for the transmission of all signals in a FlexRay cluster transmitted during the static and dynamic segment [3] [16].

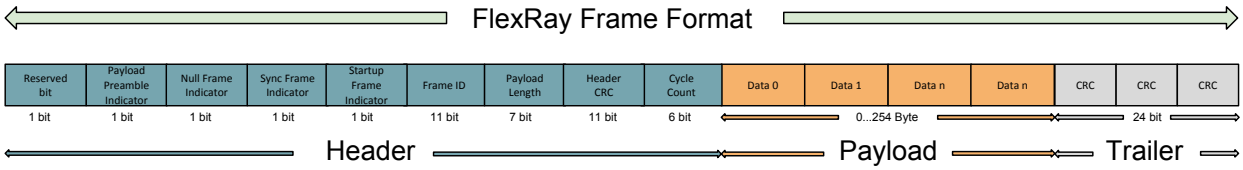

Figure 2.5: FlexRay Message Format

The header consist of a status field, frame ID, payload length, header CRC, and a cycle counter. The payload section defines the payload of the message and the trailer section contains a 24 bit CRC that secures the header and payload sections [16].

2.5.6.1 Header Segment

The header section constitutes the first 40 bits (5 Byte) of the FlexRay frame and includes the status frame, frame ID, payload length, header CRC and cycle count. The 5 bit status frame contains the reserved bit, payload preamble indicator, null frame indicator, sync frame indicator, and startup frame indicator [3].

Reserved Bit The reserved bit (1 bit) is not used and reserved for future applications. The transmitting node shall transmit a logical '0' during the bit duration, while a receiving node ignores the reserved bit [3].

Payload Preamble Indicator The payload preamble indicator (1 bit) specifies if an optional vector is present in the payload segment of the transmitted frame. For frames in the static segment, the payload preamble indicator specifies that the start of the payload segment contains a network management vector. For frames in the dynamic segment, the payload preamble indicator specifies that the start of the payload segment contains a message ID. If the payload preamble indicator bit is set to a logical '0', the payload segment contains neither a network management vector nor a message ID [3].

Null Frame Indicator The null frame indicator (1 bit) specifies that the transmitted frame is a 'null frame'. If the null frame indicator bit is set to a logical '0', then the payload segment of the belonging frame contains no valid data and all the bytes of the payload segment are set to logical '0'. If the null frame indicator is set to a logical '1' the indicator specifies that the belonging frame has data in the payload section [3].

Sync Frame Indicator The sync frame indicator (1 bit) specifies if the transmitted frame is a 'sync frame'. The sync frame is used for global network synchronisation. If the sync frame indicator bit is set to a logical '1', the transmitting frame is a sync frame and will be used by the receiving nodes for network synchronisation. If the sync frame indicator is set to logical '0', then the reviving nodes do not use the frame for any synchronisation related tasks [3].

Startup Frame Indicator The startup frame indicator (1 bit) specifies whether the transmitted frame is a 'startup frame' or not. The startup frame is transmitted by coldstart nodes and used in the startup mechanism of the FlexRay network mentioned in Section 2.5.5.2. If the startup frame indicator bit is set to a logical '1', the transmitting node is a coldstart node and the transmitted frame is a startup frame. If the startup frame indicator bit is set to a logical '0', the transmitted frame is not a startup frame. A single node can only transmit one startup frame per communication cycle [3].

Frame ID The 11 bit frame ID (values from 1-2047) with the ID '0' as an invalid ID specifies the unique ID for the belonging frame. The frame ID is used to position the frame in a slot in both the static and dynamic segment. Two nodes in a network are not allowed to transmit frames with the same frame ID on the same communication channel. The frame ID is transmitted such that the most significant bit of the frame ID occurs first in the frame ID transmission. The following bits are then transmitted in decreasing order down to the least significant bit of the frame ID [3].

Payload Length The payload length (7 bit), named the Data Length Coding (DLC) specifies the length of the words transmitted in the payload segment, divided by two. That means a 4 byte payload segment will be indicated with the value '2' in the DLC. The DLC is transmitted such that

the most significant bit of the DLC occurs first in the DLC transmission. The following bits are then transmitted in decreasing order down to the least significant bit [3].

Header CRC The 11 bit header CRC consists of a cyclic redundancy check (CRC) for the header portion of the FlexRay message. The header CRC is calculated using the sync frame indication, startup frame indicator, frame ID and the payload length. This constitutes a 20-bit input to the CRC calculation. The CRC is computed offline and not by the transmitting FlexRay communication controller, meaning that the FlexRay host controller has to calculate the header CRC for both the transmitted and the received message [3].

The header CRC is calculated in the same way for all nodes, messages and communication channels with the CRC polynomial in Equation 2.1, with an initialisation vector for the CRC register of $0x01A$ [3].

$$x^{11} + x^9 + x^8 + x^7 + x^2 + 1 = (x+1) \times (x^5 + x^3 + 1) \times (x^5 + x^4 + x^3 + x + 1) \tag{2.1}$$

In order to calculate the CRC value of the header, a CRC algorithm determines if the exclusive OR (XOR) of the CRC register and the current bit of the input data register should be applied. The input data are the 20 bits from the header portion. The current bit of the input data register refers to the most significant bit of the input data. The CRC polynomial is applied to the CRC register if the XOR function of the data and CRC register yields a logical '1'. Further, the input data register is bit shifted such that the most significant bit is shifted out and second most significant bit becomes the most significant bit. The process is then repeated for the entire length of the original input data (20 bit) [4]. The header CRC is transmitted such that the most significant bit of the CRC occurs first in the CRC transmission. The following bits are then transmitted in decreasing order down to the least significant bit [3].

The header CRC has its purpose in protecting parts of the header portion and providing a quick CRC in the header portion itself, such that no node begins to process invalid data [12].

Cycle Count The cycle count (6 bit) specifies the communication cycle iteration of the current communication cycle. All frames in the same communication cycle must have the same cycle count, which is incremented automatically by the transmitting node. The cycle count can have 64

values (0-63) which will be periodically repeated. The cycle count can also be used as a continuity index for the communication, because of its incrementing transmission. The cycle count is transmitted such that the most significant bit of the cycle count occurs first in the cycle count transmission. The following bits are then transmitted in decreasing order down to the least significant bit [12].

2.5.6.2 Payload Segment

The payload field (0-2032) can carry up to 127 words of 16-bit of useful data in the message frame. The frame ID could also carry a network management vector or a message ID if the payload preamble indicator is set to a logical '1' in the status field. The payload is transmitted such that the most significant bit of the first byte in the payload occurs first in the payload transmission. The following bits are then transmitted in decreasing order down to the least significant bit of the last byte in the payload [12].

2.5.6.3 Trailer Segment

The trailer segment of a FlexRay message contains solely a 24 bit CRC calculated by using the header segment and the payload segment. The trailer CRC protects the entire transmitted frame using the CRC polynomial in equation 2.2 for both FlexRay channels [3].

$$\begin{aligned} &x^{24}+x^{22}+x^{20}+x^{19}+x^{18}+x^{16}+x^{14}+x^{13}+x^{11}+x^{10}+x^{8}+x^{7}+x^{6}+x^{3}+x+1 \\ &=(x+1)\times(x^{11}+x^{9}+x^{8}+x^{7}+x^{5}+x^{3}+x^{2}+x+1)\times(x^{11}+x^{9}+x^{8}+x^{7}+x^{6}+x^{3}+1) \end{aligned} \tag{2.2}$$

For the 24 bit trailer CRC, the FlexRay nodes employ a different initialisation vector for each FlexRay channel. For channel 'A' the node uses the initialisation vector $0xFEDCBA$ and for channel 'B' the node uses the initialisation vector $0xABCDEF$ [3].

The 24 bit CRC is calculated by the communication controller and is verified in the same manner by receiving controllers with calculating a new 24 bit CRC based on the frame content and comparing it to the trailer segment of the received frame. This is different in comparison to the header CRC, in which the 11 bit CRC is calculated by the host controller. The 24 bit CRC

is calculated before a frame transmission and succeeding a frame reception. The 24 bit CRC is calculated in the same way as the header CRC, only that the length of the input data to the CRC algorithm is dynamic due to variable payload lengths [3].

2.5.7 FlexRay Frame Format

Each of the fields mentioned in Section 2.5.6 which is representing the logical load, are subdivided into bytes and encapsulated with a START and STOP bit, making each transmitted byte a 10 bit transmission [12]. The FlexRay byte packaging is illustrated in Figure 2.6, where the 5 bit status field and the 3 most significant bits of the frame ID are embedded within a 10 bit structure after the official start of the transmission of the frame. Further, the illustration shows the remainder of the 11 bit frame ID whereas the 8 least significant bits are transmitted after the status and 3 most significant bits utilising the same structure [12].

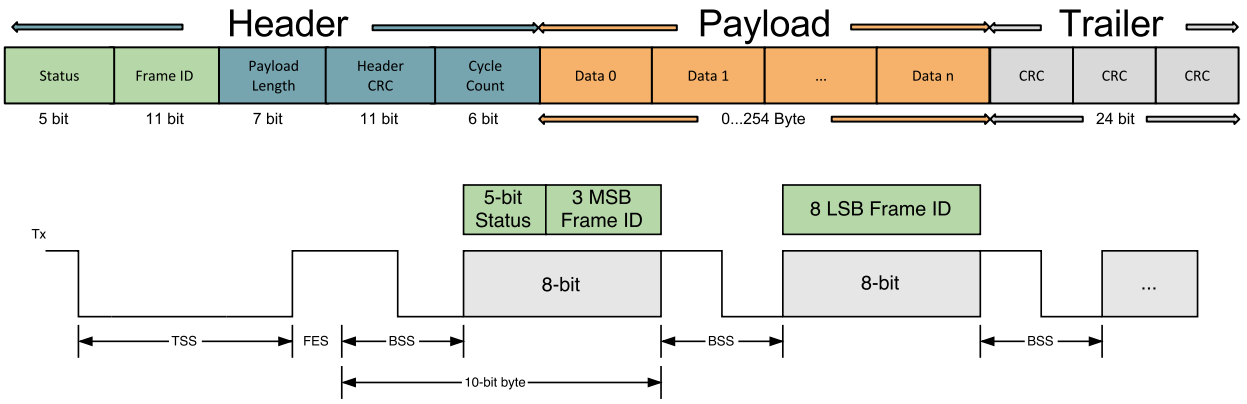

Figure 2.6: FlexRay Frame Format and Byte Packaging

For the actual bit transmission of a FlexRay frame, many precautions are added to ensure proper and secure transmission with actions taken against delays and varying frame length. The following sections will describe bit-by-bit how a FlexRay frame is structured and transmitted.

2.5.7.1 Byte Start Sequence

Each logical byte is added a START bit (logical ‘0’) at the start and an STOP bit (logical ‘1’) at the end, consequently forming a 10 bit byte of non return to zero 8N1 type. Since each transmitted byte in FlexRay is added the START and STOP bit, two successive bytes will have a STOP-START,

logical ‘1-0’ pattern in between. In FlexRay, the logical ‘1-0’ pattern is called a Byte Start Sequence (BSS) and occurs before every transmitted byte [12].

2.5.7.2 Frame Start Sequence

The start of a FlexRay frame is indicated with a logical ‘1’ called the Frame Start Sequence (FSS) and is appearing before the first BSS. The rising edge of the FSS signals the start and presence of a FlexRay frame. The combination of the FSS and the first BSS of the frame give a distinct pattern to the start of a frame with a ‘1-1-0’ logical value [12].

2.5.7.3 Transmission Start Sequence

The Transmission Start Sequence (TSS) is transmitted before the FSS in order to initiate a transmission sequence. This occurs at the start of frames in the static segment, dynamic segment and symbol segment. The length of the TSS is configured by the network designer depending on the network typologies, the distance between nodes and other physical aspects, with valid values from 3-15 bit long [12].

It takes only a single bit during the low state of the TSS for a node to become aware of an incoming frame. Due to physical delays in initiating line drivers for the receivers and transmitters in a FlexRay cluster, and/or diverting the signal through an active star, the possibility that the first edge of the transmitted frame is delayed longer than the remaining edges is very high. This effect causes the TSS signal to be perceived as shorter on the input terminals of the receivers than the actual output by the transmitting node, an effect called ‘TSS truncation’. The duration of the TSS is a globally defined FlexRay network parameter, called TSSTransmitter, which is equal in all nodes of a cluster. Therefore, due to the multiple effects, a receiving node must accept a frame when a low state occurs in the time slot reserved for the TSS with a duration from 1 bit to the defined duration + 1 bit as a valid TSS (1 to TSSTransmitter + 1). For a network with a TSSTransmitter of 10 bits, a node in the cluster must accept frames in which the TSS signal is received at a duration between 1 and 11 bits [12].

2.5.7.4 Frame End Sequence

Following the CRC in the trailer of a frame, the last byte of the FlexRay frame is closed by a specific logical '0-1', 2 bit, sequence, namely the 'Frame End Sequence' (FES). The FES are the last transmitted bits in the FlexRay frame and therefore mark the end of the frame transmission [12].

The BSS, FSS, TSS and FES are all part of the FlexRay frame transmission and persistent in every message transmitted in a FlexRay cluster, both in the static segment and the dynamic segment.

2.5.8 Static and Dynamic Slot Format in FlexRay

The FlexRay slot is the structure that consists of the FlexRay frame mentioned in Section 2.5.7 with some additional structure added. The additional structures in the FlexRay slot, apart from the FlexRay frame, are the Channel Idle Delimiter (CID), and the Dynamic Trailing Sequence (DTS) [12].

2.5.8.1 Action Point

The start of a FlexRay slot contains a 'channel idle' state, in which an action point (AP) occurs at a precise point in time. The action point refers to the exact point in (local) time at which a node performs a specific action in conformance with the local time reference. In the case of FlexRay, this refers to the time instant where Tx of the transmitter starts transmitting the frame on the medium. To ensure global time in the FlexRay network, a receiving node carries out a measurement of the time difference between the real action point and the actual reception of the signal. In this way the nodes of a FlexRay cluster can compensate for the propagation delays of the signal [12].

2.5.8.2 Channel Idle Delimiter

The CID takes place outside the FlexRay frame with logical data in order to add padding to the time between the end of the electrical frame and the end of the FlexRay slot. By design, for both the static and dynamic slot, in the ending of the structure, a field of 11 logical '1' bits is added.

Consequently, the 11 bit structure is called the CID and has the purpose to signal the end of a frame in the slot and further to clear the medium such that it can become idle [12]. The FlexRay static slot with the CID is illustrated in Figure 2.7.

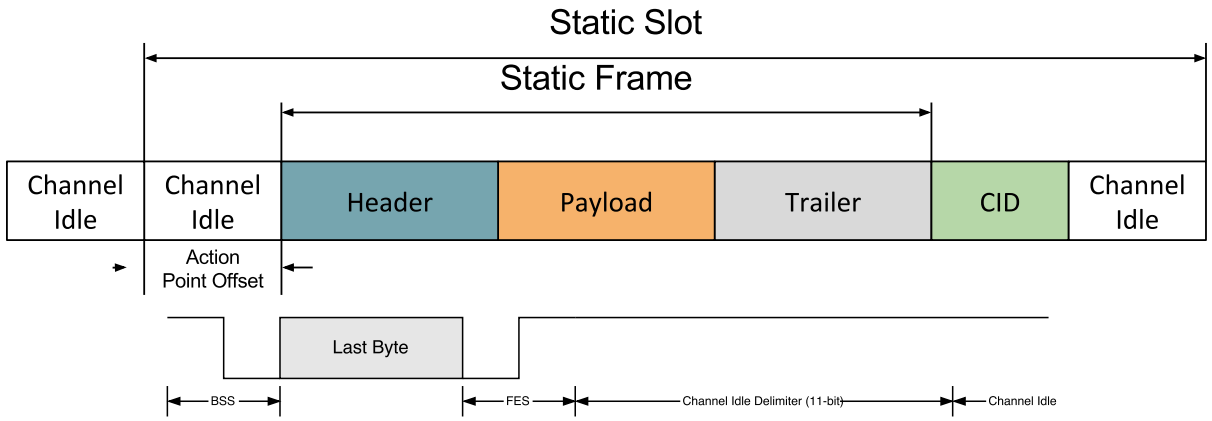

Figure 2.7: FlexRay Static Slot

2.5.8.3 Dynamic Trailing Sequence

When transmitting dynamical frames, a problem occurs due to their variable length putting it out of synchronisation with the durations depending on the action point. In order to fix this, an additional sequence is added in the transmission of dynamic frames - the Dynamic Trailing Sequence (DTS). This segment occurs between the trailer and the CID in the dynamic slot with a variable duration in order to calibrate microticks according to the durations of the minislots, and in relation to the precise position of the action point of the subsequent minislot [12].

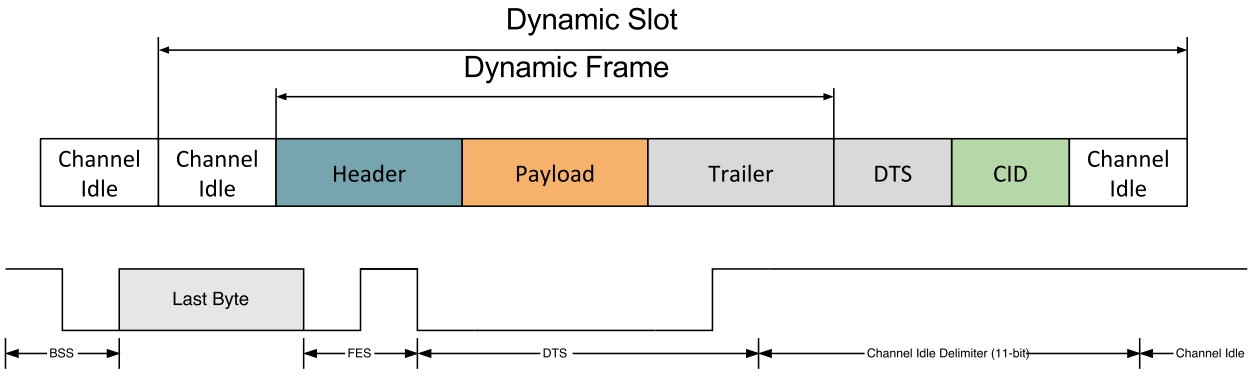

Figure 2.8: FlexRay Dynamic Slot

2.5.9 Physical Layer of FlexRay

Although, the physical layer is not explicitly defined in the protocol specification of FlexRay [3], it describes the instance of using a differential pair wired medium for network transmission. Further, it provides the designer with the flexibility to also utilise FlexRay with e.g. a fiber-optic physical layer implementation [12].

2.5.9.1 Bit Rate

The gross bit rate specified in the protocol specification of FlexRay [3] produces a 100 ns bit duration for a 10 MBit/s bit rate on FlexRay, and officially supports 2.5 MBit/s and 5 MBit/s in addition to a 10 MBit/s bit rate, with the 10 MBit/s being the maximum gross bit rate on one FlexRay communication channel. A 5 MHz signal transmission would be realised with a long varying symmetrical duty cycle sequence S, and a changing logical bit sequence B where $B = \{1,0\}$ and $S = \{B_1, B_2, B_3, ..., B_N\}$ with 100 ns bit duration [3][12].

2.5.9.2 Bit Encoding and Differential Voltage

FlexRay implements a bit encoding called 'non-return to zero' (NRZ) type in which the physical signal does not change in value during the duration of the bit after producing the signal. Both logical levels in FlexRay are represented by a dominant state while the recessive states are reserved for 'idle' mode. The physical transmission lines Bus Plus (BP) and Bus Minus (BM) are measured relative to the reference potential and the difference between the voltage potential on the transmission lines are known as the differential voltage 'V_{Bus}' in equation 2.3.

$$V_{Bus} = (V_{BP} - V_{BM}) \tag{2.3}$$

The V_{BP} and V_{BM} are the voltage potential of the BP and BM transmission lines measured relative to the reference potential. When producing a logical '1', a 'high' voltage potential is produced on the BP transmission line while a 'low' voltage potential is produced on the BM transmission line. The opposite occurs when producing a logical '0'. Therefore, the V_{Bus} will be positive for a logical '1' and negative for a logical '0'. There is no data transmission in 'idle' mode and no node is in a 'low power mode'. The low power modes, idle_lp are achieved when the BP

and BM transmission lines are grounded with pull-down resistors, resulting in a voltage potential of 0 V for the entire FlexRay communication channel [12]. The bit representations are illustrated in Figure 2.9.

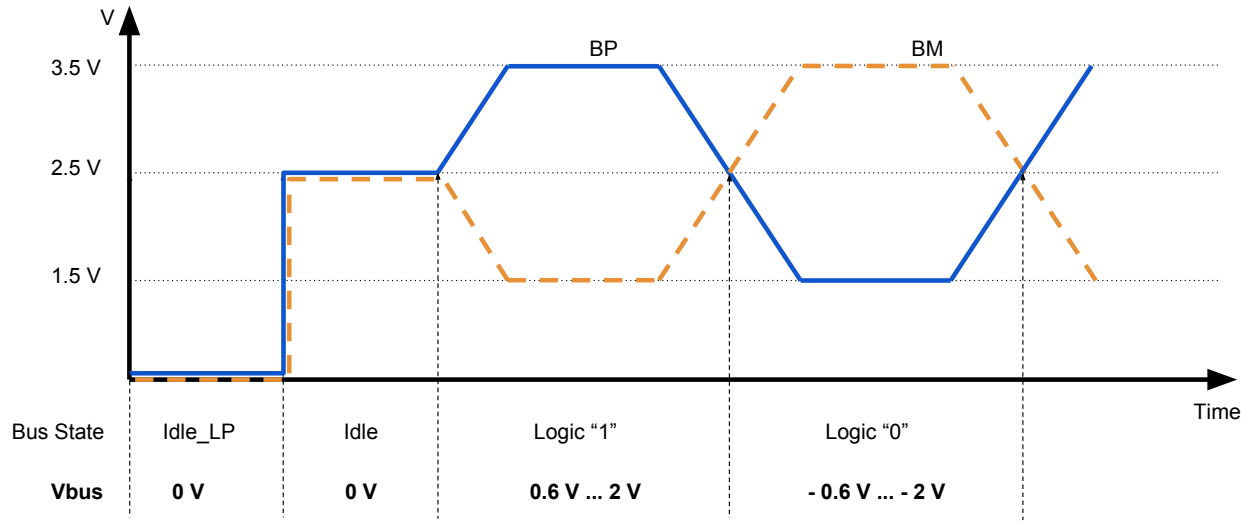

Figure 2.9: Physical Bit Representation of ‘Differential Pair Wire’ with NRZ Bit Encoding

2.5.9.3 Asymmetric Delays

A symmetric power stage is almost impossible in physical applications, as an integrity fault between the transmitted and received signal is caused in the rise and fall times of signals propagated throughout the network. The protocol specification of FlexRay [3] states a tolerable limitation of the asymmetric delays at 4 ns and 5 ns for the transmitter and receiver respectively. Further, the tolerance is increased to 8 ns for the receiver when an active star is used for signal delegation [12].

2.5.10 FlexRay Node Architecture

For the general architecture of a FlexRay node, four major sections are implemented; a host microcontroller, protocol manager, line driver and an optional bus guardian. The communication cycle, communication controller and onward are managed by the protocol manager, while a host microcontroller implements the application management [12]. The general architecture with external and internal connections is shown in Figure 2.10.

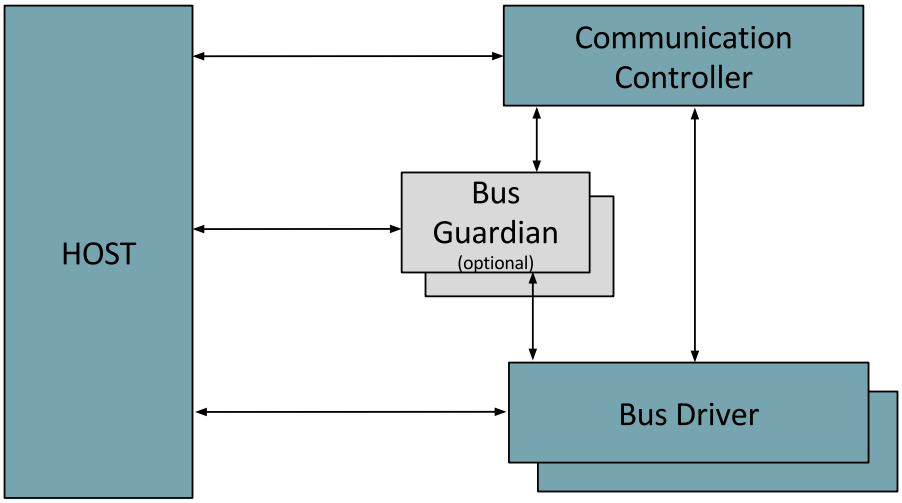

Figure 2.10: Node Architecture

Bus termination (impedance-matching) is implemented in FlexRay nodes in order to minimise drift inductance and reduce asymmetrical delays. The termination is usually supplemented with common mode coils and should be implemented symmetrically between the nodes in order to match the typical 80 to 110 Ohm wiring impedance in a FlexRay network [12].

Bus Guardian The bus guardian is an optional feature in FlexRay and its main objectives are to prevent a node from getting access to the medium in a wrong time slot and to block a 'babbling idiot' in the network. The bus guardian is aware of the exact communication timing and is synchronised with the communication controller [12].

2.5.11 In-Cycle Control

Due to the exact timing and the structure of the static segment, FlexRay is well suited for executing 'in-cycle control' where data is transmitted during the early stages of the static segment and a response is given in the same cycle. The Figure 2.11 illustrates an example where gyroscope values are transmitted from an ECU (ECU 1) in the static segment of the communication cycle, allowing an additional ECU (ECU 2) to process the gyroscopic values and transmit an update signal in the same communication cycle [8].

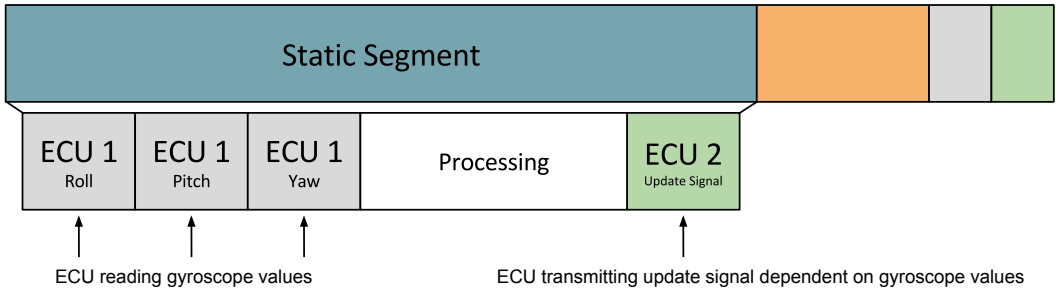

Figure 2.11: In-Cycle Control Showed in the Static Segment

2.5.12 Packet Sniffing a FlexRay Network

The high security demands and growing complexity of X-by-wire systems led to the introduction of time-triggered architectures for data transmission. The time-triggered architectures operate a static transmission schedule and consequently reduce the complexity of the execution flow for the receivers. Further, the decrease in execution flow complexity has led to an increase in configuration complexity in which FlexRay requires multiple 10s of configuration parameters. The theoretical amount of unique configurations of FlexRay communication is more than 10^{48} configurations [1].

According to [15], a standard open source tool for sniffing packets/frames in a FlexRay network is not available (in 2016). The available generic tools for this purpose are proprietary and costly, and additionally usually require the FIBEX file (configuration and communication setup) for the FlexRay network in order to function. The same conclusion was drawn in the pre-project of this paper [7]. In order to packet sniff a FlexRay network without the FIBEX file, the bit rate of the network has to be known. Optimally, also the duration of the communication cycle and the start and length of the static and dynamic segment should be known [15].

In the paper *'Automatic Parameter Identification in FlexRay based Automotive Communication Networks'* [1], a method for automatically identifying FlexRay network parameters is suggested and could be implemented in a packet sniffing FlexRay system in order to also provide parameter information.

2.6 FlexRay Test Network

The two-node FlexRay test network was developed during the pre-project of this paper, "*On-board Communication Systems in Vehicles - FlexRay*" [7]. The network consists of two TMS570LS1227 microcontrollers implemented on separate development boards with external FlexRay bus drivers. The bus drivers perform according to protocol specification 2.1 of FlexRay [3]. For the FlexRay configuration and communication part, a test software issued by Texas Instruments is implemented on both nodes, each having a separate role in the communication as node 'A' and node 'B' [7].

The function of the test network was validated in [7] and in this paper the test network will be used to validate results from a FlexRay analysis software, as the FlexRay configuration parameters are given in the example software. Figure 2.12 shows the two node FlexRay test network.

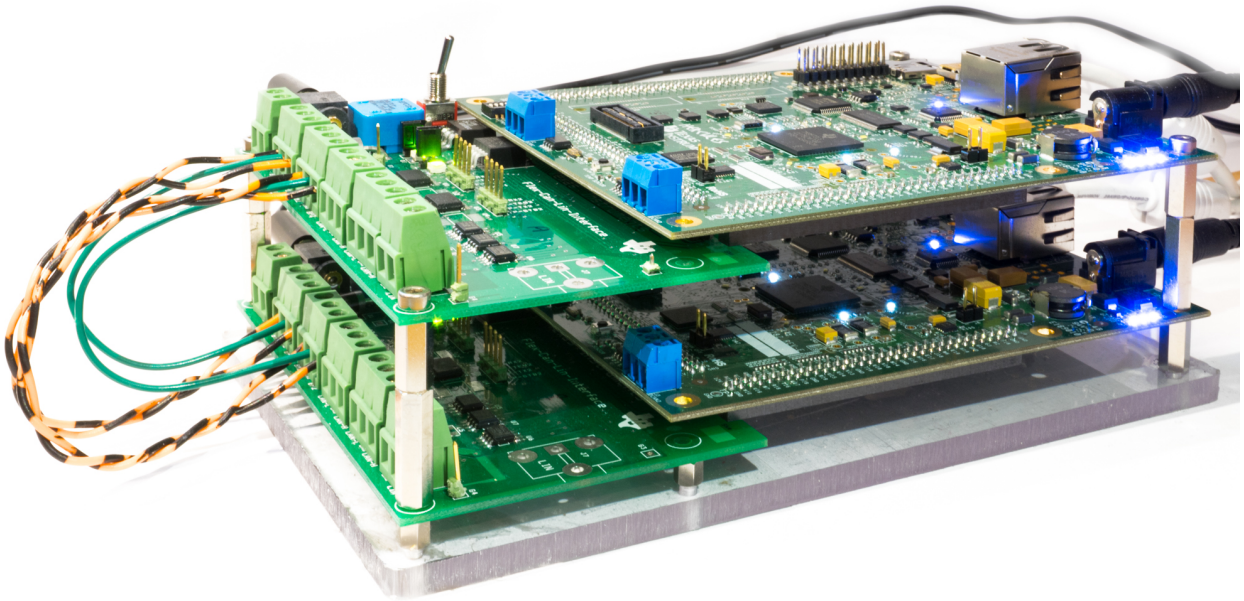

Figure 2.12: FlexRay Test Network

2.7 Automotive Data Security

A modern vehicle consists of multiple ECUs which transfer data between each other and are connected trough one or multiple communication networks. If an adversary is able to achieve physical access to the vehicle's CAN bus, the adversary is able to intercept, manipulate and broadcast messages [5]. Further, an adversary that has physical access to the vehicle, can perform a 'man-in-the-middle' attack on any of the communication protocols [14].

According to [17], a malfunction or a malicious on-board real-time communication protocol which operates driving commands such as steering and braking, can lead to severe danger for the affected occupants and surrounding environment. The potential for impact by an attack on the different communication networks is in the area of what is shown in Table 2.1 [17].

Table 2.1: Impact Potential of Selected Automotive Communication Protocols

Communication Protocol	MOST	LIN	CAN	FlexRay
Endangerment	Low	Low	High	Very high

2.7.1 Previous Research

Researchers at the University of California San Diego and the University of Washington demonstrated how they manipulated and controlled on-board systems trough the CAN bus. The attack resulted in the systematic control of multiple of the vehicle's components; including brakes, heating and cooling system, engine control, lights, radio, locks and instrument panel cluster. The manipulation included various techniques in order to gain control over some of the vehicle's functions, mainly packet sniffing, packet fuzzing and ECU flashing as described next [9].

Packet Sniffing Packet sniffing involves observing the data traffic of the on-board network in order to determine how the ECU communicate between each other [9].

Packet Fuzzing Packet fuzzing simply involves sending random messages into the network in order to cause unexpected behaviour and uncover security vulnerabilities within the system [9].

ECU Flashing This technique involves 'reverse engineering' the software for some of the vehicle's ECUs in order to add additional functionality rather than using the excising software capabilities in the vehicle [9].

Following this research, two researchers [10] implemented the same techniques as in [9], however, without the need of physical manipulation in the vehicle. The research exposed security flaws connected to the telematic unit and in the connection to the cellular network. In this way, the same attacks as used in [9], could be implemented by remotely reprogramming the telematics control unit. Therefore, multiple of the vehicle's functions could be controlled through remote access to the CAN network of the vehicle. The attack worked on any car of the same model in the entire US, which presumably led to a massive recall of the targeted vehicle [10].

2.7.2 Man-in-the-Middle-Attack

Wireless communication interfaces are beyond the purpose of this paper and the focus will be set on an adversary connected to the wired connection of the communication protocol. In order to establish the kind of manipulation that is presented in this paper, passive connection is not an option as the original message needs to be manipulated and cannot conflict with potential injected messages. Further, passive connection to the FlexRay protocol defies any means of the protocol specification as all nodes and network communication are configured by a global design, engineered by the network designer, specified to the targeted applications. Therefore, the focus of this paper will be put on a specific form of attack, namely the Man-in-the-Middle (MitM) attack.

Figure 2.13 illustrates a man-in-the-middle attack of just one node/ECU of an arbitrary automotive communication network (e.g CAN). The node under attack is the single node on the one side of the adversary 'ECU subjected to adversary' or 'Node under attack' (NUA). The adversary could also be situated in other configurations, attacking more than one node, and multiple adversaries could be implemented in a network. A man-in-the-middle attack could also have full authority over the network if an adversary is placed in front of every node in the network except one (randomly chosen). Adding an additional controller or system to the network communication will have an influence on the network. An adversary will introduce a delay in the network as the transmission signal propagates through, and is processed by, the adversary [14].

Figure 2.13: A Man-in-the-Middle Attack on an Automotive Communication Network

According to [14], for CAN and LIN networks, the delay caused by the adversary should be close to a passive forwarding of data. The adversary has to read a single symbol in order to manipulate and forward it. Further, for FlexRay and MOST, due to the use of time division multiple access schemes for medium access, the delay is given by $delay = n \times cycletime$ (with n as a positive integer) as a consequence of the TDMA scheme [14].

The functionality of any given ECU in a network, whereas the delay of the adversary is existent, is highly application specific and can therefore be limited by the manipulation. Assuming that the communication protocol only used the security and timing features inherent in the protocols, and that sub-sequential transfers had no relations with each other, the delay approach mentioned above would be suitable for any of the mentioned automotive communication networks [14].

Due to the event based messaging structure inherent in CAN, where no data is expected to be received at a specific point in time, the functionality of a node can be expected not to limit the original functionality. For a node in a FlexRay cluster, the delay might induce much greater limitations to both the node and the network. As FlexRay is typically used for timing critical applications and timing critical communication. Further, for the TDMA scheme for bus access to the medium, the timeout intervals have to be small and closely monitored for such applications [14].

2.7.3 AUTOSAR End-to-End Communication Protection

AUTOSAR[2] has implemented features to further secure information between electronic control units communicating over automotive networks.

[2]Automotive Open Systems Architecture (AUTOSAR) is a global partnership that provides an industry standard for the field of software architecture for Original Equipment Manufacturers (OEMs) and suppliers in the automotive industry [12]

Since the AUTOSAR system functions on a higher layer than the FlexRay communication, the protection is application related and suggesting security features, which are implemented in the payload of the transmitted frame. The protection is addressing failure modes as defined by ISO DIS 26262 for the communication, e.g. insertion, corruption, incorrect sequence, timing faults, addressing faults. For the data protection, an 8 bit CRC is transmitted in the payload calculated over the frame ID, a counter and the rest of the payload. The counter is also transmitted as a part of the payload of the frame [6].

Chapter 3

Investigating the Test Vehicle

3.1 Electronic Systems Overview

The vehicle used for the FlexRay investigation is a Mini One 2016 model. The vehicle is used in a driving simulator and therefore mounted on a motion platform in order to simulate an external environment and different driving conditions. The vehicle mounted on the motion platform is shown in Figure 3.1.

Figure 3.1: Mini One

In order to create a notion of what to expect when analysing the FlexRay communication in the vehicle, it is necessary to establish an overview of the electrical systems in the vehicle. Due to proprietary solutions in both hardware and software and the general secrecy of the automotive

industry, little information is available about the electrical systems of the vehicle. Further, no information is available regarding the FlexRay network configuration or the data in the network. However, connector configurations for the different electronic control units, gateways, electrical sensors and so forth are publicly available on the manufactures' web pages through payed access or licensed proprietary diagnostics tools. The connector information can be used in order to establish an overview of external connections to the different ECUs.

The 'BMW ISTA Rheingold' diagnostics tool was used in order to retrieve information about the ECU connector, and each ECU in the vehicle was inspected for bus connection. The connector information displays the pin-layout for a specific connector of an ECU, and would indicate 'FR_BP' and 'FR_BM' for the FlexRay bus plus and bus minus transmission lines respectively when the units are connected to FlexRay.

The electrical systems are listed in table 3.1. LIN modules are not mentioned as they consist of a large amount of modules (e.g. the light operating unit and exterior mirrors) and are not assumed to be of significant importance for the purpose of this paper. The connector information displays multiple (4) CAN networks; a single MOST/Ethernet network and a single FlexRay network. A 'central gateway module' connects the multiple networks and is physically the same unit as the 'body domain controller'. Regarding the BMW/MINI terminology the engine control unit is the Digital Motor Electronic (DME) of petrol engines. As the airbag control unit (Crash Safety Module), the dynamic stability control, the engine control unit, and the electromechanical power steering are all connected to the same FlexRay network, it can be assumed that safety-critical information is transmitted throughout the network.

3.2 Access to FlexRay Transmission Lines

In this paper, the point of access for the FlexRay network in the vehicle is established through a direct connection on the FlexRay BP and BM transmission lines in the ECU for the EPS system. This is due to easy access as the wheels of the vehicle are disconnected and the ECU and the electric servomotor are connected closely to the steering wheel shaft connection to the electromechanical steering system, available in the front left wheel arch. The ECU for the EPS system has three main connections; FlexRay (BP and BM) and ignition voltage in addition to battery voltage.

Table 3.1: Electrical Control Units and Network Connection, Mini One

Communication Network	System	Acronym
K-CAN2	Parking Manoeuvring Assistant	PMA
K-CAN2	Roof Module	FZD
K-CAN3	Headlight L/R	
PT-CAN	Instrument Cluster	KOMBI
PT-CAN	Electronic Transmission Management	EGS
PT-CAN	Digital Motor Electronics	DME
K-CAN4	Information Computer	HU-B
K-CAN4	Integrated Automatic Heating System	IHKA
K-CAN4	Radio Module	
K-CAN and PT-CAN	Body Domain Controller	BDC-CAS
K-CAN and PT-CAN	Central Gateway Module	ZGW
Ethernet	Radio Module	
Ethernet	Body Domain Controller	BDC-CAS
Ethernet	Central Gateway Module	ZGW
FlexRay	Digital Motor Electronics	DME
FlexRay	Crash Safety Module	ACSM
FlexRay	Electromechanical Power Steering	EPS
FlexRay	Dynamic Stability Control	DSC
FlexRay	Body Domain Controller	BDC-CAS
FlexRay	Central Gateway Module	ZGW

The ECU for the EPS contains an embedded 32 bit Texas Instruments TMS570LS microcontroller similar to the one used in the FlexRay test network in Section 2.6.

The petrol engine of the vehicle is removed and simulated through multiple sensors in order to make the rest of the electronic system "think" that the engine is running. Since multiple of the original systems are replaced by external sensors and manipulated CAN frames, one can expect the data transmission on the multiple bus systems to contain error messages. Further, the FlexRay frames may indicate errors in different systems. Therefore, the network communication might be different in a fully functional vehicle.

Chapter 4

FlexRay Analysis Software

The FlexRay Analysis Software (FAS) is developed in order to retrieve frame content and timing information from an existing FlexRay network while only knowing the bit rate in advance. Additionally, the frames have to comply with the FlexRay frame format in Section 2.5.7 and both the 11 bit and 24 bit CRCs must be calculated and compared to the header CRC and trailer portions of the transmitted FlexRay frame.

The input data for the FAS is sampled directly from the FlexRay transmission lines with a passive bus connection to the original network. The sample data must contain the exact sample time and the measured analogue voltage for each transmission line, with a sample rate faster than the Nyquist frequency for the bit duration in the FlexRay network. A lower sampling threshold was not tested for the FAS, as the sampling rate of the equipment used in this paper was much faster (10-20 times) than the duration of a 100 ns bit.

The sample data from the sampling equipment is stored in a simple format (.txt or .csv) such that the software could be applied to all kinds of analogue samples from a FlexRay network with an appropriate sampling time and resolution.

The MATLAB code for FAS is provided in Appendix B, Section 10.3.

4.1 Analogue Samples to Digital Signal

4.1.1 Analogue Samples

The first line of the analogue sample file contains the voltage and time delimiters for the specific sample file, which are used to ensure proper values and timing in the software. The rest of the file is then loaded into a matrix M where $M \in \mathbb{R}^{n\times 3}$ and n is the number of samples in the file. The first column of M contains the time-stamp and the second and third column contain the analogue voltage of the BP and BM signal. A plot of the analogue samples for one FlexRay frame is shown in Figure 4.1.

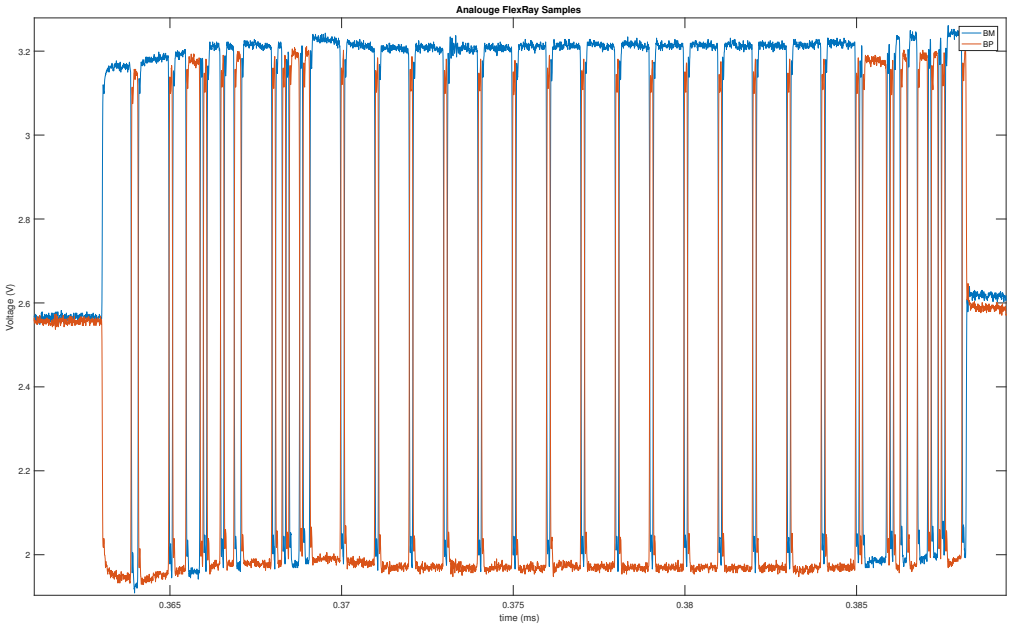

Figure 4.1: Analogue Representation of a FlexRay Frame

4.1.2 Defining Voltage Thresholds

The voltage thresholds for logic '1', logic '0', idle and idle_LP are defined according to Section 2.5.9.2 in terms of equation 2.3. Meaning that for a logic '1', the V_{Bus} voltage is defined as 0.6 V to 2.0 V, whereas V_{Bus} is the voltage difference between the BP and BM lines respectively. For a logic '0' the V_{Bus} is defined as -0.6 V to -2.0 V. The idle state will produce a V_{Bus} voltage of 0 V, however,

this will have to be addressed in the physical world as the BP and BM voltages are not exactly similar in the idle states. Therefore, an experimental threshold of -0.1 V to 0.1 V is used in order to detect the idle states.

The V_{Bus} signal is simply calculated using equation 2.3, then $\vec{V_{Bus}} = \vec{BP} - \vec{BM}$, where $\vec{V_{Bus}} \in \mathbb{R}^{n\times 1}$ and n is the number of samples in the file. Further, the $\vec{V_{Bus}}$ will be the main vector when converting to a digital signal. Since the FlexRay transmission lines produce the idle states and the NRZ type bit signal, there is a total of four different bus states present on the bus lines. The different states must be addressed in order to develop a fully digital signal. The FAS makes no distinction of the idle and the idle_lp states as only the frame content and the timing information are of importance.

4.1.3 Classifying the Signal with Voltage Thresholds and Averaging

4.1.3.1 Voltage Thresholds

The $\vec{V_{Bus}}$ is classified by using the thresholds in order to reduce the amount of unique sample voltages. All states are currently preserved and are defined as in Table 4.1.

Table 4.1: Bus State Voltages

Bus States	Idle Standby	Idle	Low	High
Voltage	0 V	2.6 V	1.8 V	3.3 V

The classifying method used for the different states compares the current value of the $\vec{V_{Bus}}$ to the analogue thresholds in Section 2.5.9.2 and appoints one of the states from Table 4.1 to the current sample in the signal. The idle standby state and the idle state are inseparable by the $\vec{V_{Bus}}$, however, can be detected when the actual analogue voltages of the transmission lines are 0 V. Figure 4.2 shows the classified V_{Bus} vector plotted over the sample time. The main 'High', 'Low' and 'Idle' states are classified, however, a problem occurs in the flanks of the bit signals as they are not within any of the tolerances for the defined states. This is further shown in Figure 4.3, as the $\vec{V_{Bus}}$ is plotted over a time period of about 1600 ns (16 full bit duration's on a 10 Mbps transmission rate). The flanks then have to be corrected such that they conform with the correct classified $\vec{V_{Bus}}$ signal.

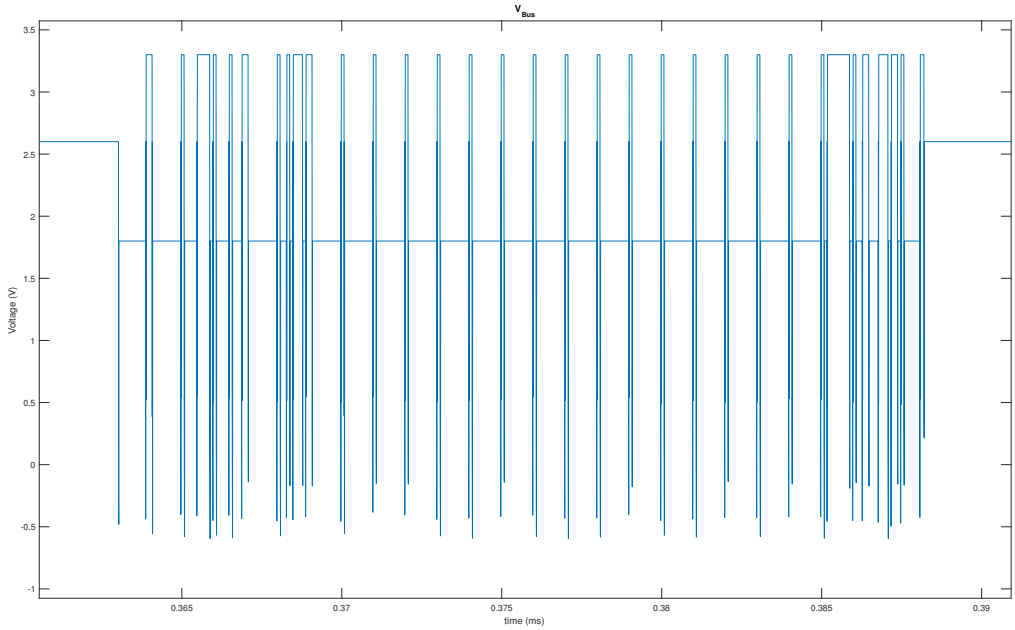

Figure 4.2: $\vec{V_{Bus}}$ Classified with Tolerances

When classifying the $\vec{V_{Bus}}$ with the threshold, all the $\vec{V_{Bus}}$ values that are not corrected with the threshold are registered such that they can be addressed by its index in the V_{Bus} vector. During the shift of a logical bit state on the bus, the $\vec{V_{Bus}}$ value can lay inside the threshold for the idle state. In this case the classifying will register this as an idle state when it is actually in transition from one bus state to another. Therefore, both the unchanged $\vec{V_{Bus}}$ values and some of the idle states have to be corrected.

4.1.3.2 Averaging

For correcting the unidentified samples, averaging is used whereas the V_{Bus} vector is investigated for a change in value. When a change in value occurs, a finite horizon average of the $\vec{V_{Bus}}$ values forward in time is calculated and compared to a voltage threshold for the average value.

For a 5 ns sample rate signal, a horizon of 11 samples was used in order to calculate the average value. Threshold for the average value was set to 2 V, meaning that an average value of the horizon of a voltage greater or equal to the threshold would set the current value of $\vec{V_{Bus}}$ to 'High' and an average voltage less than the threshold would set the current value of the $\vec{V_{Bus}}$ to

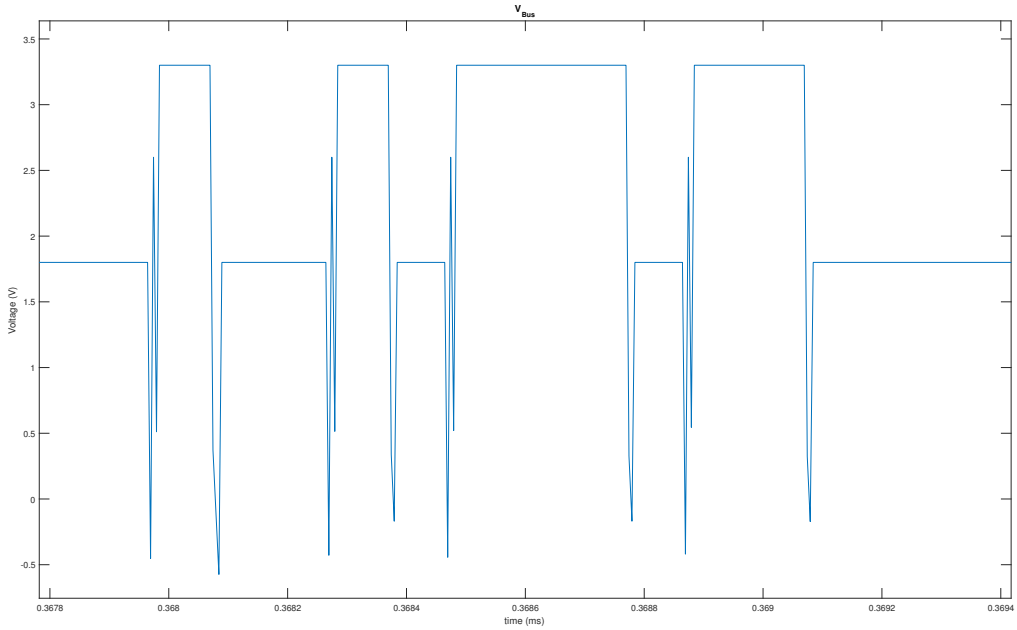

Figure 4.3: $\vec{V_{Bus}}$ Classified with Tolerances, 16 Bit Durations

'Low'.

An average value calculated over an horizon of 11 samples contains about 8 samples of either a bus state of 'High' 'Low', or 'Idle' and there is a maximum of one 'Idle' sample during the bit change. The other values that are not corrected are in the range that is not covered by the thresholds given in section 2.5.9.2. Therefore, a threshold for the average value can be estimated. Table 4.2 shows the average voltages calculated over an 11 sample horizon. The median of the lowest average voltage for a sample that should be corrected to 'High', and the highest average voltage for a sample that should be corrected to 'Low' is about 2 V and therefore, the threshold for the average value is set to 2 V.

Table 4.2: Average Voltages Calculated with a Horizon of 11 Samples

8 Samples	**1 Sample**	**1 Sample**	**1 Sample**	**Average Voltage [V]**
High/Low	Idle	0 V	0 V	**2.64/1.55**
High/Low	Idle	-0.59 V	-0.59 V	**2.53/1.43**
High/Low	Idle	0.59 V	0.59 V	**2.74/1.65**
High/Low	0 V	0 V	0 V	**2.40/1.30**
High/Low	-0.59 V	-0.59 V	-0.59 V	**2.24/1.15**
High/Low	0.59 V	0.59 V	0.59 V	**2.56/1.47**

In order to determine if the sample should be corrected, the sample index is compared to the index for the uncorrected samples. Further, the program checks if all the indexed uncorrected samples are corrected and are occupied by a bus state. The $\vec{V_{Bus}}$ then contains only three unique values; 'High', 'Low' and 'Idle'. The result of the averaging is shown in Figure 4.4, where a single FlexRay frame is filtered by threshold and corrected with averaging.

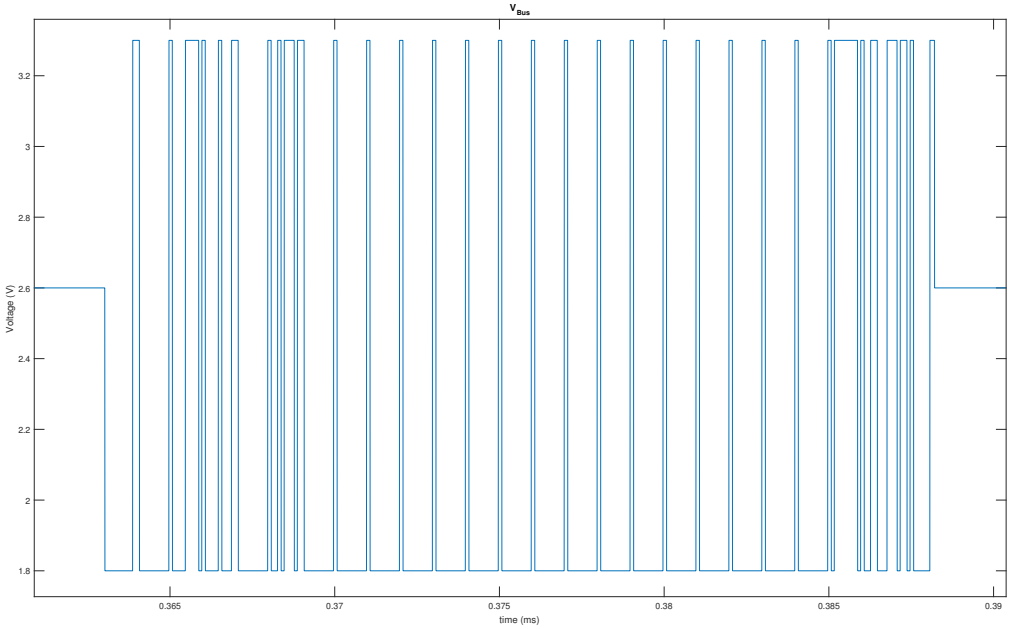

Figure 4.4: $\vec{V_{Bus}}$ Values of a FlexRay Frame After Classification

4.1.4 Digitising the FlexRay Signal

In order to create the bit signal (a stream of 'ones' and 'zeros') of the sample data, the $\vec{V_{Bus}}$ now has to be converted to a digital representation. In order to reduce the amount of samples the software has to handle, the nonzero values of the derivative of the $\vec{V_{Bus}}$ ($d\vec{V_{Bus}}$) are used in comparison with the original time-stamp of the sample value. The $d\vec{V_{Bus}}$ signal will contain values that originate from the change of a bus state. The derivative values from the change of bus states are listed in Table 4.3. In order to create a bit signal only using the two logical states '1' and '0', all bus states apart from 'Low' are changed to 'High'. The channel idle period will then be represented as a long bit stream of logical 'ones'. The 'transmission start sequence', which

indicates the start of a frame transmission to the receivers, has a pattern of 3-15 'zeros' at the start of the frame. Frames transmitted in both the static and dynamic segment are padded with the 'channel idle delimiter', which is an 11 bit signal of 'ones' at the end of the frame in the slot. Therefore, changing the idle bus states to logical 'ones' will not change the content of a frame or the method of decoding the signal. Changing the idle bus states to logical 'zeros' would make it impossible to identify the length of the TSS.

Table 4.3: Values of $d\vec{V}_{Bus}$ During a Change in Bus State

Change in Bus State	**Value of the Derivative**	**Logical Value of Current Sample**
High to Low	Low - High = -1.5 V	**0**
High to Idle	Idle - High = -0.7 V	**1**
Low to High	High - Low = 1.5 V	**1**
Low to Idle	Idle - Low = 0.8 V	**1**
Idle to High	High - Idle = 0.7 V	**1**
Idle to Low	Low - Idle = -0.8 V	**0**

Figure 4.5 shows the $\vec{V_{Bus}}$ (blue) compared to the $d\vec{V}_{Bus}$ (in orange) of one FlexRay frame.

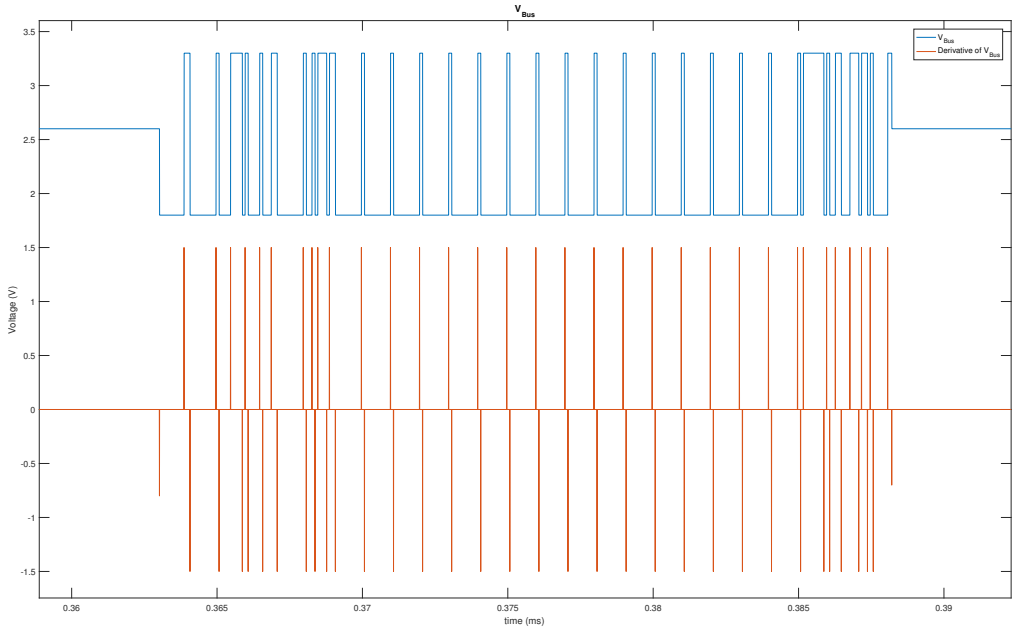

Figure 4.5: $\vec{V_{Bus}}$ and $d\vec{V}_{Bus}$ of a FlexRay Frame

Figure 4.6 illustrates how a FlexRay frame is represented when changing the idle bus states to logical 'ones'.

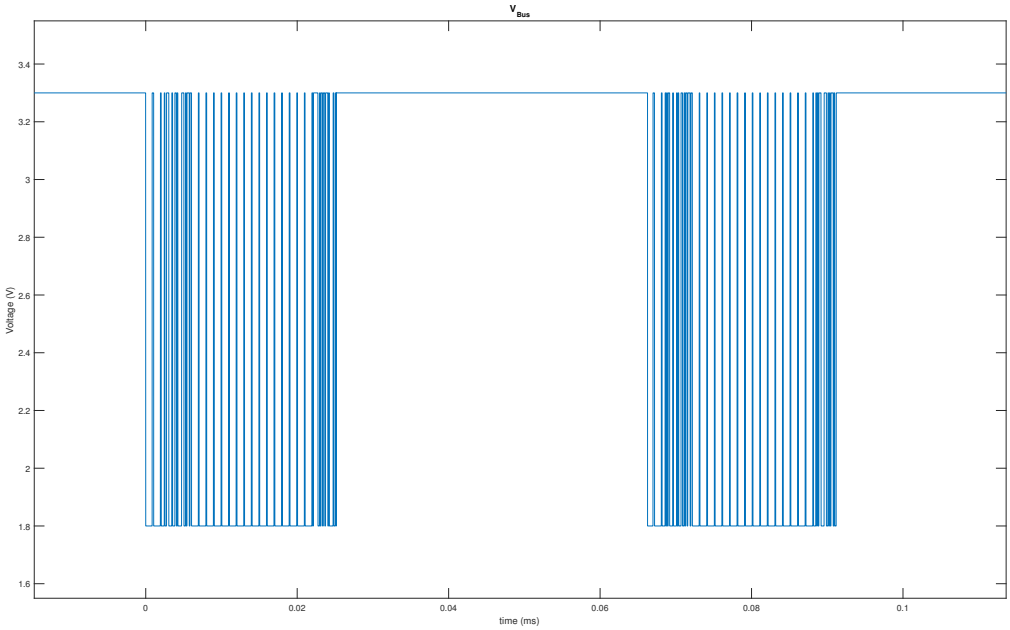

Figure 4.6: $\vec{V_{Bus}}$ Represented with Only Logical 'Ones' and 'Zeros', Two FlexRay Frames

The vector with the bit signal now contains the logical values calculated from the derivative, and the original time-stamp of the sample value. This dramatically decreases the size of the vector, as only the change in value is saved, and not the entire sample. With the use of the non-return to zero type bit encoding, the actual logical bit values can be calculated using the time-stamp, the bit signal from the derivative, and the bit rate of the FlexRay network. A 10 MBit/s bit rate produces a 100 ns bit duration. Therefore, if two sequential samples of the $d\vec{V}_{Bus}$ have a logical value of '1' and '0' with time-stamp 0 ns and 400 ns respectively, the samples would represent a logical '1-1-1-1' bit signal. When converting the analogue values to logical 'ones' and 'zeros', the actual change in the voltage on the medium is the point in time used for a change in the logical values. The reason for using the change of the value and not the actual timing for when the value reaches a threshold for either 'High' or 'Low' is due to the timing in the bit transmission in FlexRay, as the other method would be a slight offset in time while the duration of the bit would be the same.

The method used for calculating the sequential stream of bit values then calculates the time between every consecutive element in $d\vec{V}_{Bus}$, which will then indicate the logical bit value of the analogue signal and the duration of the bit value. The time is calculated based on the time-

stamp in the analogue sample data compared with the time delimiter in order to be expressed in nanoseconds. In order to divide the duration into separate bits, and not as a measurement of how long the physical bus signal has been 'High' or 'Low', the duration of the bit value is divided by the bit duration of the FlexRay network and rounded up.

The result is a matrix A where $A \in \mathbb{R}^{2\times n}$ and n is the number of times a bit value changes in the sample. The matrix A contains the current bit value, and the number of consecutive bits of the same bit value after the change of a bus state. The matrix is illustrated in table 4.4 and gives the following bit stream: '**1-10-00000000-10-00011110**'. The bit stream is divided for illustration purposes in order to show the FSS as the first bit, followed by two 10 bit byte transmissions including its 2 bit BSS. Thus, the bit stream contains the symbol segment and frame ID of a specific FlexRay frame.

Table 4.4: Logical Bit Values and Number of Consecutive Bits

Element in matrix:	k	k + 1	k + 2	k + 3	k + 4	k+ 5
Logical bit value:	1	0	1	0	1	0
Number of consecutive bits:	2	9	1	4	4	1

4.1.5 Identifying Slots and Frames

4.1.5.1 Converting Bit Stream into Slots

The matrix A is then converted to a bit stream that consists of the actual logical values of the sampled signal. Since the bit stream is calculated using the persistent timing in FlexRay, and the threshold values for the FlexRay signals, the bit stream will then indicate what any receiver on the network is experiencing.

In order to retrieve and validate the information in all frames in the sample data, the bit stream is split when a pattern equal to the CID sequence padded with a logical 'zero' at the end occurs in the bit stream. Consequently, the bit stream will then be divided into n bit streams where n is the number of frames in the sampled signal. The information from a FlexRay message will then include all the logical bits starting with the first bit of the TSS, until the last bit before the start of the TSS for the successive frame. Therefore, the FlexRay slot information is also included.

4.1.5.2 Converting Slots into Frames

For each FlexRay slot, the FlexRay frames are retrieved and converted into their correct segment using the FlexRay frame format from Section 2.5.7. The process of dividing the bit stream from the frame into its respective segments proceeds as described in the following paragraphs.

Separate frame and channel idle bit The CID sequence and its following channel idle bits are stored in the variable *ChannelIdle_Bits*. Since the bit stream contains all the bits from the start of the frame until the start of the subsequential frame, the bit stream can be separated by the CID sequence.

Separating the messages by the CID results in two vectors; one that contains only static frames or the dynamic frames including the DTS, and a second vector that contains the CID and the number of idle bits until the start of the successive frame. *ChannelIdle_Bits* will then indicate two times the 'channel idle' time, when multiplied with the bit duration and subtracted the 11 bit CID.

Calculate the DTS For messages transmitted during the dynamic segment, the end of the FlexRay message contains the additional dynamic trailing sequence. The length of the DTS is calculated over the number of logical 'zeros' before the CID. If a DTS is present in a FlexRay message, it will indicate that the frame is transmitted during the dynamic segment of the communication cycle.

TSS, FSS, BSS, FES After removing the channel idle time, the CID, and the DTS from the message, the rest of the message will consist of the entire FlexRay frame.

The start of each bit stream is investigated for a '0-1-1' logical bit pattern, whereas the logical '1' marks the start of the frame. The zeros before the FSS are then the 3-15 long TSS. Further, in order to confirm that the program is evaluating the start of the frame, the subsequent bits after TSS and FSS are compared with the logical '1-0' BSS sequence. The end of the bit stream is compared with the logical '0-1' FES.

All the bits following the FSS in the message are transferred into matrix K where $K \in \mathbb{R}^{n\times 10}$ and n is the number of 10 bit bytes of NRZ 8N1 type in the current FlexRay frame as mentioned

in Section 2.5.7. The matrix K is then checked for a correct BSS by comparing the first two bits of each row in K to the BSS '1-0' pattern, in order to determine a correct division of the bit stream. Further, the actual bytes of K are transferred to the matrix B where $B \in \mathbb{R}^{n\times 8}$ and n is the number of bytes in the frame that have logical content.

4.1.6 Identifying and Validating FlexRay Frame Parameters

The matrix B is now consisting of the header, payload and trailer portion of the FlexRay message as shown in Figure 2.6, which is the correct FlexRay message format.

Header Portion The first 40 bits (5 byte) of the matrix B ($B \in \mathbb{R}^{n\times 8}, n = \{1,2,3,4,5\}$) contain the header portion of the FlexRay frame, and each parameter is retrieved for its belonging bits as shown in table 4.5. Meaning that the FAS then retrieves the status, frame ID, payload length, header CRC and cycle count parameters of the FlexRay frame.

Table 4.5: Retrieving Header Parameters from $B, B \in \mathbb{R}^{n\times 8}$

Header Parameter	n	Bits
Status (5-bit)	1	1..5
Frame ID (11-bit)	1,2	6..8,1..8
Payload Length (7-bit)	3	1..7
Header CRC (11-bit)	3,4,5	8,1..8,1..2
Cycle Count (6-bit)	5	3..8

Payload Portion The payload portion of the FlexRay frame follows the header portion. Since the header portion has a static length of five bytes, the bytes that contain the payload portion can be calculated using the payload length from the header portion of the current FlexRay frame. The DLC indicates the number of bytes in the payload sequence, divided by two ($DLC = \frac{PayloadLength}{2}$). Therefore, the payload portion is retrieved from the matrix B for all bits when $n = 6..(5 + 2 \times DLC)$ and DLC ≥ 1, e.g for a frame with a DLC of 2, the belonging bytes of B would be when $n = \{6,7,8,9\}$ and $n \in B$. If the transmitted frame has a DLC of 0, then the payload segment is not a part of the frame.

Trailer Portion The trailer portion of the FlexRay frame is the last three bytes of the matrix B and is retrieved from B using the same logic. The trailer portion consists of the 24 bit CRC for the FlexRay frame as calculated by the transmitting node.

4.1.6.1 Validating Header CRC

In order to validate the header portion of the FlexRay frame, a new 11 bit CRC for the header portion is calculated and compared with the header CRC of the received FlexRay frame following the format mentioned in Section 2.5.6. The hexadecimal value of the CRC polynomial from Equation 2.1 for an 11 bit CRC is equivalent to $0x385$.

The input to the header CRC algorithm is a 20 bit segment containing, from the left, the sync frame indicator, startup frame indicator, frame ID and payload length.

The pseudo-code for the implementation of the header CRC is given in Algorithm 1, with the only external input being the respective parts of the header portion used in the CRC calculation. The algorithm returns the respective 11 bits of the CRC register. The MATLAB implementation of the 11 bit header CRC is shown in appendix B, Section 10.1.

Result: Calculates the 11 bit header CRC

Initialise: header (20 bit), CRC_Register (0x1A), CRC_Polynomial (0x385), length (20);

Initialise temporary registers: CRC_Next(0), header_temp(0), reg_temp(0);

while *length* **do**

length - -;

header ≪ 1;

header_temp = MSB of header;

reg_temp = MSB of CRC_Register;

if *header_temp XOR reg_temp* **then**

CRC_Next = 1;

else

CRC_Next = 0;

end

CRC_Register ≪ 1;

if *CRC_Next* **then**

CRC_Register = CRC_Register XOR CRC_Polynomial;

end

end

Return: CRC_Register && 0x7F

Algorithm 1: 11 Bit Header CRC

4.1.6.2 Validating Trailer CRC

In order to validate the entire FlexRay frame, a 24 bit CRC is calculated by the FAS and compared to the value of the trailer in the received frame, following the format mentioned in Section 2.5.6. The CRC is calculated for both initialisation vectors for the CRC register, $0xFEDCBA$ for channel 'A' and $0xABCDEF$ for channel 'B' with the CRC polynomial from Equation 2.2 equivalent to $0x5D6DCB$

The 24 bit CRC takes the entire header and payload sections as input data for the CRC algorithm which functions almost in the same manner as the one used for the 11 bit CRC in Algorithm 1. Since the input data is significantly larger than in the 11 bit CRC, and even a frame with no payload would be at least 40 bit long due to the 5 bytes of the header section, the input

data cannot be stored in a 32 bit integer on its own. Further, only input data where the frame has a DLC of less than 1 can be stored in a 64 bit integer. Therefore, a shift register is implemented for the input data were a 32 bit integer is used as the shift register. The input data is converted to a string containing all the bit values of the header and payload sections with the first bit of the header section located leftmost in the string. The CRC algorithm then evaluates the most significant bit of the 32 bit shift register and when shifting out the most significant bit, the least significant bit of the shift register is taken from the appropriate position in the input data string. The CRC algorithm is processed for the entire length of the input data string and returns the respective 24 bits of the CRC register for validation. In order to validate the frame, the CRC register for both channel 'A' and channel 'B' is compared to the value of the trailer for the current frame. The MATLAB implementation of the 24 bit trailer CRC is shown in appendix B Section 10.2.

The output from the FlexRay analysis software displays the parameters from all static and dynamic frames including DTS, CID and the channel idle time until the successive frame. Additionally, the validity of the frames is checked with the correct FlexRay frame format and both the header and trailer CRC are calculated by the FAS in order to approve the frame content and simultaneously confirming the functioning of the analysis. In order to sample a FlexRay network, the bit rate of the network has to be known in advance such that the correct bit duration can be configured in the FAS.

Chapter 5

FlexRay Communication Analysis

In order to provide an overview of how the FlexRay communication is structured in the test vehicle, the FlexRay analysis software explained in Chapter 3 is first tested on the FlexRay test network where we have insight into the FlexRay configuration and communication parameters. Then the same testing is done for the FlexRay network in the test vehicle.

At an early stage in the software development process, a 'Saleae Logic 8' logic analyser (100 MS/s digital sampling) was used in an experiment for capturing the differential signal from FlexRay. A high frequency operation amplifier comparator circuit was developed in order to convert the differential wire FlexRay signals to TTL (0 to 5V) digital signals in order to sample the bus communication, as the logic analyser uses standard 5 V TTL for digital signal sampling. However, due to noise close to the reference voltage, the digital representation of the FlexRay signals showed inconsistencies between the BP and BM lines, meaning that one of the transmission lines would have a change in value where the other line was static for a multiple of bit durations.

Instead, the physical signals in the FlexRay network were sampled with a Picoscope 4225, which is an automotive grade PC oscilloscope with a 20 MHz frequency operating 12 bits at 400 MS/s. This implies that the oscilloscope will take a maximum of 40 samples of a 100 ns FlexRay bit duration.

5.1 FlexRay Analysis in Test Network

5.1.1 The Communication Cycle

In order to identify the communication cycle of the transmissions in the test network, the sample data is analysed with respect to the specific segments in the communication cycle. Figure 5.1 shows a sample of the BM line expressed in voltage (y-axis) vs. time (x-axis). A time larger than the typical 5 ms for the duration of the communication cycle is chosen in order to plot the entire cycle. The sample data is taken from FlexRay channel 'A' on the test network. Here a recurrent pattern is recognised where two consecutive frames are transmitted followed by a short idle time period and then two longer frames followed by a long idle time period. The time between the first frame after the long idle period and the next occurrence of this message is calculated to be 5.6 ms. It is therefore assumed that the four frames belong to the same communication cycle and the long idle time consists of the NIT segment which ends the communication cycle. The plot in Figure 5.1 has a time interval of about 20 ms and will, therefore, contain just under four full communication cycles with a duration of 5.6 ms.

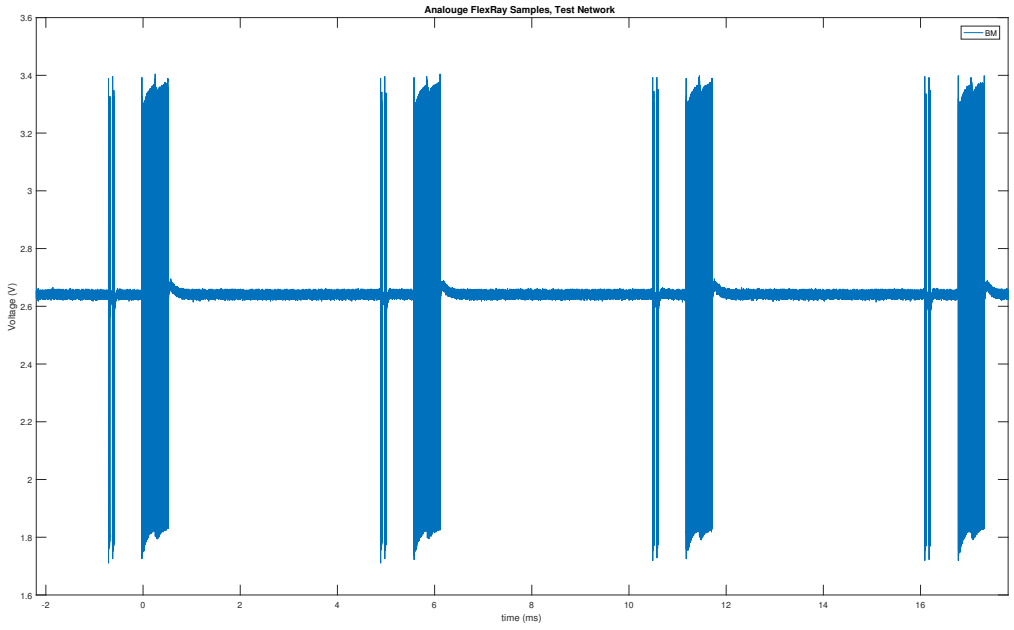

Figure 5.1: FlexRay Sample Data from BM Line

Further, the Figure 5.2 shows the start of the communication cycle and the frames in the static segment or the static and dynamic segment of the sample data as in Figure 5.1.

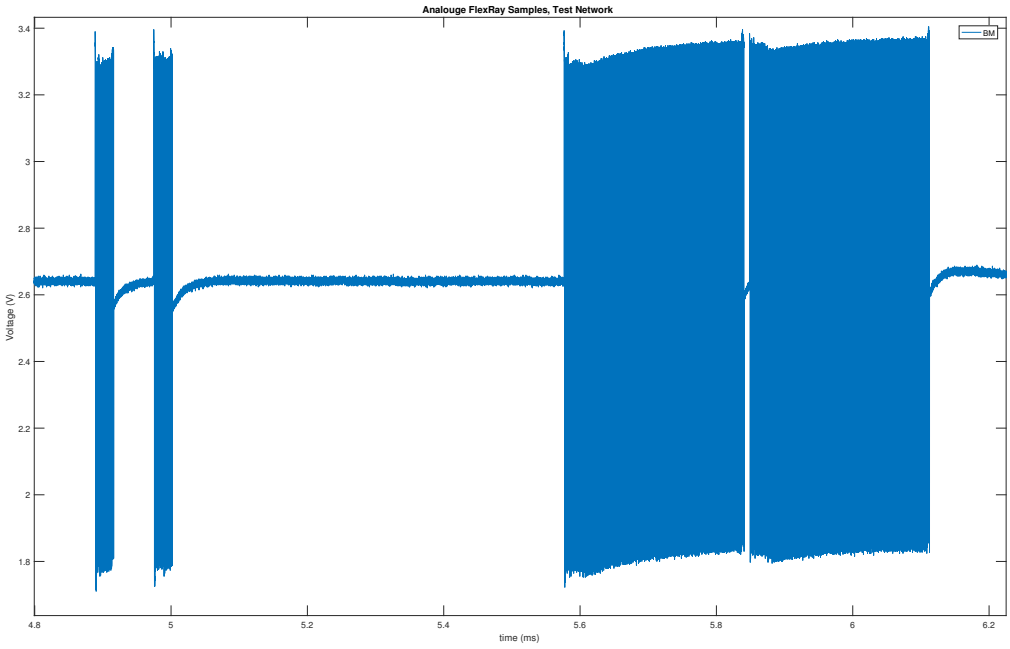

Figure 5.2: FlexRay Sample Data from BM line, Start of Cycle

In order to further analyse the communication the analogue sample data is processed with the FlexRay analysis software from Chapter 3. The first frame of the sample data which is assumed to be the first frame of the communication cycle is presented in Figure 5.3 and results from the FAS.

5.1.2 The Static and Dynamic Frames

The FAS shows that there is a reoccurring pattern of four consecutive frame transmissions in the sample data with iterating cycle count for each pattern, which is further supporting the communication cycle duration theory. Table 5.1 shows the header content of the four messages in the communication cycle, in which the 11 bit header CRC calculated by the FAS is identical to the header CRC in the original FlexRay frame. The 'Frame Number' refers to the occurrence of the frame in respect to the sample data. Due to the static frame length requirement and the position of the static segment in the communication cycle it is clear that at least the frame ID $0x1$

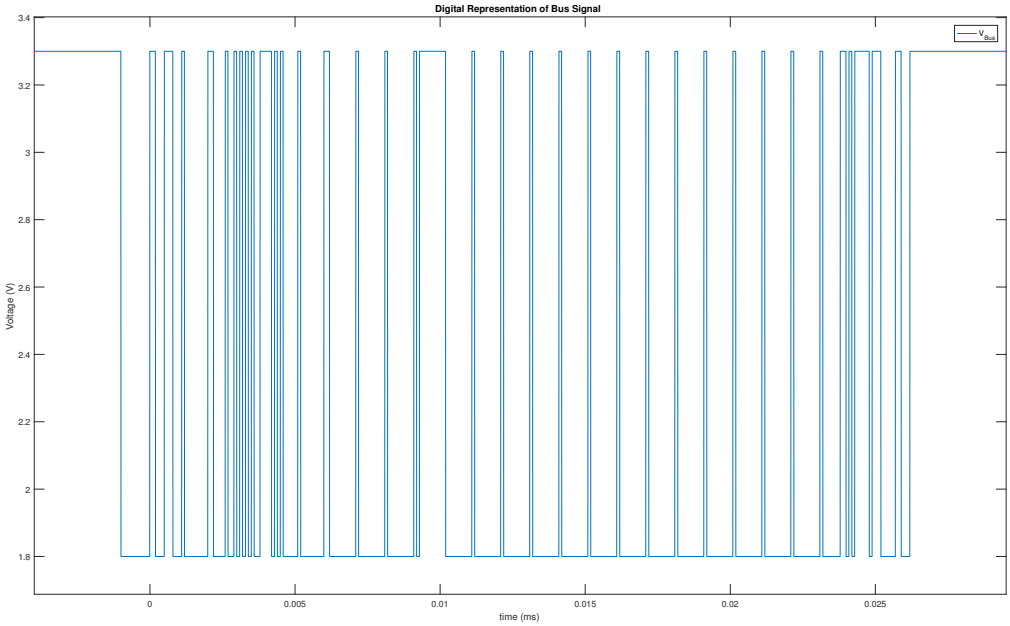

Figure 5.3: Digital Representation of $\vec{V_{Bus}}$ for Frame '1' In Test Network

belongs to the static segment. However, it is highly likely that frame ID $0x2$ is also transmitted during the static segment due to its location and the idle time after the transmission of frame $0x2$. When inspecting the 'status' parameter $0x7 = 0b111$ of Frame ID $0x1$ and $0x2$ this implies that for the two frames, the null frame indicator bit, sync frame indicator bit and the startup frame indicator bit are set to a logical '1'. The status parameter confirms that both ID $0x1$ and $0x2$ are a part of the startup and synchronisation process of the network and are, therefore, located in the static segment. The frames with IDs $0x9$ and $0xA$ are then located in the dynamic segment of the communication cycle. The null frame indicator is set for all frames in the current communication cycle which implies that all frames have data in the payload section of the frames.

The DLC parameter refers to the payload length of the frame, meaning that the frames transmitted in the static segment have a payload length of 18 bytes while frame ID $0x9$ and $0xA$ have a payload length of 254 bytes ($0x7F = 0d127$).

Table 5.2 shows the payload and trailer content of the four frames in the communication cycle. The 'CRC Valid CH A/B' parameter indicates whether the calculated 24 bit CRC for channel 'A' or channel 'B' is identical to the trailer of the transmitted FlexRay frame. The calculated 24 bit CRC will only be identical to the trailer portion of the transmitted FlexRay frame for either

Table 5.1: Frame Header Content, Test Network

Frame Number	Status	Frame ID	DLC	CRC	Cycle Count	Header CRC Valid
1	0x7	0x1	0x9	0x29E	0x20	√
2	0x7	0x2	0x9	0x81	0x20	√
3	0x4	0x9	0x7F	0x5E1	0x20	√
4	0x4	0xA	0x7F	0x7FE	0x20	√

channel 'A' or channel 'B' due to the different initialisation vectors for the different channels as mentioned in Section 2.5.6.3.

The calculated CRC is identical to the transmitted CRC for all four messages in the communication cycle, and the frame content is therefore valid. The payload of the frames is not analysed as it is a test network with no external input that influences the frame content.

Table 5.2: Payload and Trailer Content, Test Network

Frame ID	Payload (hex)	Trailer	CRC Valid CH A/B
0x1	01 00 00 00 FF 00 ... 00	0x6FB0C	√/-
0x2	78 56 34 12 21 43 65 87 00 ... 00	0x5D687E	√/-
0x9	FF 00 00 00 FF FF 00 00 FF FF FF 00 FF ++	0xD8B79C	√/-
0xA	FF 00 00 00 FF FF 00 00 FF FF FF 00 FF ++	0xFC7091	√/-

Further, Table 5.3 is showing the timing parameters as calculated by the FAS. For the last sampled frame, the 'Idle Until Next Frame' is not calculated as the FAS has no information about the start of the sub-sequential frame. The TSS is perceived as by a receiver and is therefore subjected to TSS truncation. The TSS length is equal for all frames in the test network in the current sample. The length of the DTS is 0 bits for all frames in the sample data, which is always the case for frames transmitted in the static segment. However, the DTS is used in the dynamic segment to compensate for the variable payload length such that the frame will end in accordance with a minislot. The two dynamic frames in the network do not contain a DTS and are therefore aligned with a whole number of minislots.

Table 5.3: Timing Parameters, Test Network

Frame ID	TSS length	Idle Until Next Frame	DTS
0x1	10 bits	57.6μs	0 bits
0x2	10 bits	573.5μs	0 bits
0x9	10 bits	6.8μs	0 bits
0xA	10 bits	-	0 bits

The length of the static slot can be calculated from the information given by the FAS using the format from Section 2.5.8. The static slot contains the entire frame, the CID and the channel idle time before the frame and after the CID. The length of the static frame is the total duration of the TSS, the FSS, the header portion, the payload portion, the trailer potion and the FES. The 'channel idle' duration of the FlexRay slot is calculated using the number of bits that follows the CID until the start of the frame transmission for an immediately successive frame (frame ID + 1).

The lengths of the header (5 byte) and the trailer (3 byte) are static and the length of the payload is given by the FAS. The content of the static slot is shown in Table 5.4 and results in a 860 bit duration of the static slot. Two static slots are occupied in the network and the time between the last static slot and the first dynamic will fit at most six whole static slots. Consequently, the number of static slots in the network configuration is at the minimum two and at the maximum eight slots.

Table 5.4: Length of Static Slot (in bits), Test Network

Channel Idle	**TSS**	**FSS**	**Header**	**Payload**	**Trailer**	**FES**	**CID**	**Channel Idle**
288	10	1	5×10	9×2×10	3×10	2	11	288

It was not possible to determine the duration or the number of minislots as the dynamic slots are of a different size than the minislots, and the network transmission consists of only two immediate successive dynamic frame transmissions. The only conclusion that can be made regarding the minislot is that the duration of a minislot is smaller than the size of the smallest dynamic slot in the dynamic segment. The dynamic slots in the test network are equal in size and the channel idle time is calculated using the time difference between the immediate successive dynamic frame transmissions of frame ID $0x9$ and $0xA$ (3.4 μs).

The total duration of the dynamic slot is calculated in the same manner as the duration of the static slot with the additional DTS included and results in a dynamic slot duration of 271.2 μs for the frames with a DLC of 127. Therefore, the minislot has to be smaller than this value. For the symbol segment of the communication cycle, no symbol was observed during the sampling, and the duration of the symbol segment is therefore not specified. Further, since the dynamic segment can hold minislots after the end of the dynamic slot with the highest frame ID, the start of the NIT segment is ambiguous and defined as the time after the ending of the last sampled dynamic slot. Further, when comparing the maximum duration for the minislot and the

minimum duration of the dynamic segment, a minimum number of minislots can be calculated.

The results from the FAS regarding the FlexRay configuration parameters in the test network are shown in Table 5.5. The 'Length of a minislot' is most likely dramatically overestimated as the duration is usually around 1 to 4 μs.

Table 5.5: Estimated FlexRay Configuration Parameters, Test Network

Reference	**Value**
Macrotick Cycle length	5600 μs
Number of Static Slots (n)	2≤ n ≤ 8
Duration of a Static Slot	86 μs
Number of Minislots (n)	n ≥ 2
Length of a Minislot (d)	d ≤ 271.2 μs
NIT Start (t)	t ≥ 1224 μs
Symbol Start (t)	t ≥ 1224 μs
Payload Length in Static Segment (DLC)	9 (18 Bytes)
Number of TSS Bits (b)	b ≥ 10 bit

5.1.3 Validation with FlexRay Node Configuration

The results from the FlexRay analysis of the test network are compared to the actual network design of the test network with an emphasis on confirming the data in Table 5.1, Table 5.2 and Table 5.5. The FlexRay network is set to a 10 MBit/s bit rate with a macrotick and microtick duration of 1 μs and 25 ns respectively. Table 5.6 shows some of the FlexRay configuration parameters in the test network. The length of the communication cycle is set to be 5600 MT which is identical with the 5.6 ms displayed in Table 5.5. Further, both the duration of the static slot and the number of static slots are equal in the FlexRay configuration when compared to the value showed in the same table. The 18 byte payload length in the static segment and the duration of the TSS complies with the values found with the FlexRay analysis software. Additionally, the node configuration is showing that 24 message buffers (frame IDs) are available with the buffers 0 to 3 assigned to the static segment and the remaining buffers are assigned to the dynamic segment. Only four message buffers are configured in the nodes and are displayed in Table 5.7.

The message buffer configuration in the test network showed in Table 5.7 is identical to the results of the FAS. The software found the same frame ID´s, payload length and status content as the message buffer configuration. However, the FAS was not able to identify the origin of the

Table 5.6: Initialisation of FlexRay Nodes, Test Network

Reference	Value
Macrotick Cycle length	5600 MT
Number of Static Slots	8
Duration of a Static Slot	86 MT
Number of Minislots	346
Length of a Minislot	4 MT
NIT Start	on MT 2887
NIT Start	on MT 2173
Microtick Duration	25 ns
Macrotick Duration	1 μs
Payload Length in Static Segment (DLC)	9 (18 Bytes)
Number of TSS Bits	10 bit

frames, as a FlexRay frame does not contain any content that would identify the transmitting node. The message buffer configuration also shows that the messages transmitted during the static segment of channel 'A' are also transmitted during the static segment of channel 'B', however, no data is transmitted during the dynamic segment of channel 'B'.

Table 5.7: Message Buffer Configuration, Test Network

Message Buffer	Channel	Origin	DLC	Sync Frame	Startup Frame
1	A/B	Node A	9	√	√
2	A/B	Node B	9	√	√
9	A	Node A	127	-	-
10	A	Node B	127	-	-

Table 5.8 shows the payload of the message buffers in the test network. The nodes use a static payload, which is identical to the payload found by the FlexRay analysis software. The first byte of the payload of the corresponding FlexRay frame is transmitted such that the least significant byte of the 32-bit in 'Data 1' is transmitted first.

Table 5.8: Payload Configuration, Test Network

Message Buffer	Data 1,2,3..n
1	0x00000001, 0x000000FF
2	0x12345678, 0x87654321
9	0xFF, 0xFFFF, 0xFFFFFF, 0xFFFFFFFF, 0xFFFFFF00, 0xFFFF0000
10	0xFF, 0xFFFF, 0xFFFFFF, 0xFFFFFFFF, 0xFFFFFF00, 0xFFFF0000

5.2 FlexRay Analysis in the Test Vehicle

The access to the transmission lines for the FlexRay network in the test vehicle are directly established on the BP and BM lines connected to the ECU for the Electromechanical Power Steering (EPS) system, available in the front left wheel arch. The wheel arch and the test equipment are shown in Figure 5.4 and the EPS ECU with the test probes connected is shown in Figure 5.5, note that the EPS ECU is not connected to the vehicle in the figure as this picture was taken during later testing. The EPS ECU is connected to the vehicle in the analysis unless mentioned otherwise.

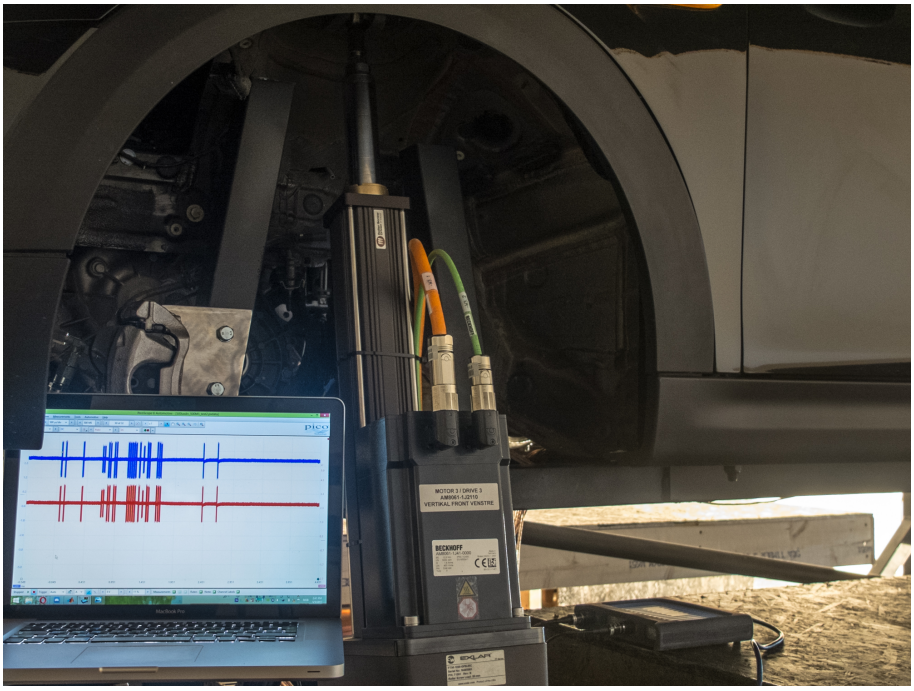

Figure 5.4: Access to the Transmission Lines Through Front Wheel Arch

5.2.1 The Communication Cycle

The sample data from the FlexRay network of the test vehicle shows a reoccurring pattern of 23 frames followed by a portion where frames appear dynamically. Figure 5.6 shows a plot of the sample data with almost three full communication cycles, where the orange line marks one full communication cycle and the green line marks the reoccurring pattern of 23 frames. The duration of the communication cycle is exactly 5 ms, calculated by using the time difference from

Figure 5.5: Electronic Control Unit for EPS System

the first transmitted frame in the communication cycle and the first transmitted frame of the successive communication cycle .

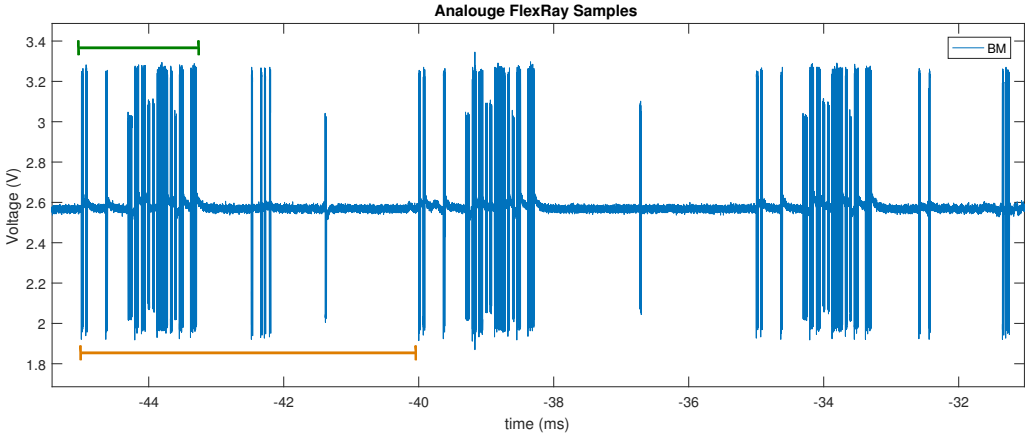

Figure 5.6: FlexRay Analogue Sample Data, BM Line

5.2.2 The Static and Dynamic Frames

When analysing the sampled data with the FAS, the reoccurring pattern of 23 frame transmissions followed by a dynamic frame transmission is confirmed. Further, the frames in the successive

communication cycle appear with the same frame IDs and an iterated cycle count. Figure 5.7 shows the reoccurring pattern of 23 frame transmissions of an arbitrary communication cycle as calculated by the FAS. Table 5.9 shows the header content of the same communication cycle with the additional dynamic frames.

The frames with ID $0x3D$, $0x40$, and $0x46$ have the sync frame and startup frame indicator bit set to a logical '1' in the status segment of the frames ($0x3 = 0b11$) which indicates that the frame with ID $0x46$ and all preceding frames in the same communication cycle are transmitted during the static segment of the cycle. Further, all the 23 frames have the same static payload length of 16 byte which are enforced for all frame transmissions in the static segment. The two frames preceding the frame ID $0x46$ have a different payload length and are a part of the dynamic segment of the communication cycle.

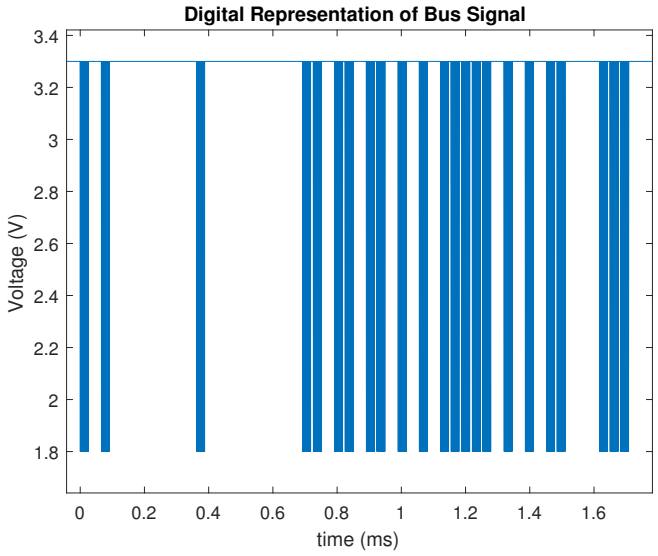

Figure 5.7: Digital Representation of FlexRay Signal

The 24 bit CRC in the trailer portion of the frame is identical to the calculated CRC with initialisation vector for channel 'A', for all 25 frames in Table 5.9, indicating that the FAS verified all 25 frames in the sample data and the data transmission occurs on channel 'A' in the FlexRay network. The payload of the frames is dynamic and the null frame indicator bit is frequently used between communication cycles for messages where there is data transmitted in the payload section of the frame. The payload of the frames is further analysed in Section 5.2.4.

Table 5.10 shows the timing parameters as analysed by the FAS for the same data sample as in

Table 5.9: Header Content, Vehicle Network

Frame Number	Status	Frame ID	DLC	CRC	Cycle Count	Header CRC Valid
1	0x0	0x13	0x8	0x60D	0x1D	√
2	0x0	0x15	0x8	0x233	0x1D	√
3	0x0	0x1E	0x8	0x401	0x1D	√
4	0x4	0x28	0x8	0x45E	0x1D	√
5	0x0	0x29	0x8	0x5AB	0x1D	√
6	0x4	0x2B	0x8	0x641	0x1D	√
7	0x0	0x2C	0x8	0x38A	0x1D	√
8	0x4	0x2E	0x8	0x60	0x1D	√
9	0x4	0x2F	0x8	0x195	0x1D	√
10	0x0	0x31	0x8	0x259	0x1D	√
11	0x0	0x33	0x8	0x1B3	0x1D	√
12	0x4	0x35	0x8	0x58D	0x1D	√
13	0x4	0x36	0x8	0x792	0x1D	√
14	0x0	0x37	0x8	0x667	0x1D	√
15	0x4	0x38	0x8	0x781	0x1D	√
16	0x4	0x39	0x8	0x674	0x1D	√
17	0x0	0x3B	0x8	0x59E	0x1D	√
18	0x3	0x3D	0x8	0x683	0x1D	√
19	0x4	0x3F	0x8	0x24A	0x1D	√
20	0x3	0x40	0x8	0x417	0x1D	√
21	0x0	0x44	0x8	0x4E0	0x1D	√
22	0x0	0x45	0x8	0x515	0x1D	√
23	0x3	0x46	0x8	0x29	0x1D	√
24	0x4	0x7C	0x4	0x19A	0x1D	√
25	0x4	0x83	0x4	0x293	0x1D	√

Table 5.9. The duration of the TSS differs between the frames and has a maximum length of 9 bit in the current sample data. Therefore, the global TSS duration is greater or equal to 900 ns. For the messages transmitted during the dynamic segment of the current communication cycle, two frame IDs are recorded, namely ID $0x76$ and $0x83$ with both a DTS duration of 34 bits.

The parameters in the static slot of two different frames are shown in Table 5.11. Both frames have a frame in the successive message buffer slot in the communication cycle and different TSS durations. When summing up the parameters in the static slot shown in the table, the duration of the static slot equals 330 bits for both examples, which is a static slot duration of 33 μs. The highest frame ID in the static segment is $0x46 = 0d70$ which implies that there are at least 70 static slots configured in the FlexRay network in the vehicle. Further, the duration of the static segment it at least 70×33 μs = 2310 μs.

In order to further determine the duration of the static and dynamic segments, more samples had to be taken such that additional dynamic transmissions (if occurring) could be analysed. Additional sampling over a longer time, and under different circumstances will most likely introduce new dynamic frames.

A dynamic slot is transmitted in multiples of minislots, rendering the minislot to be smaller

Table 5.10: Timing Parameters, Vehicle Network

Frame ID	TSS length	Idle Until Next Frame	DTS
0x13	9 bits	40 μs	0 bits
0x15	7 bits	270.7 μs	0 bits
0x1E	8 bits	303.9 μs	0 bits
0x28	9 bits	6.7 μs	0 bits
0x29	9 bits	39.7 μs	0 bits
0x2B	7 bits	6.9 μs	0 bits
0x2C	7 bits	39.9 μs	0 bits
0x2E	7 bits	6.9 μs	0 bits
0x2F	7 bits	39.8 μs	0 bits
0x31	9 bits	39.6 μs	0 bits
0x33	9 bits	39.7 μs	0 bits
0x35	7 bits	6.9 μs	0 bits
0x36	7 bits	6.9 μs	0 bits
0x37	7 bits	6.9 μs	0 bits
0x38	7 bits	6.9 μs	0 bits
0x39	7 bits	39.9 μs	0 bits
0x3B	7 bits	39.9 μs	0 bits
0x3D	9 bits	39.7 μs	0 bits
0x3F	7 bits	6.9 μs	0 bits
0x40	7 bits	105.9 μs	0 bits
0x44	7 bits	6.9 μs	0 bits
0x45	7 bits	6.7 μs	0 bits
0x46	9 bits	920 μs	0 bits
0x7C	7 bits	54 μs	34 bits
0x83	7 bits	-	34 bits

Table 5.11: Length of Static Slot (in bits), Vehicle Network

Frame ID	CH. Idle	TSS	FSS	Header	Payload	Trailer	FES	CID	CH. Idle
0x28	33.5	9	1	5×10	8×2×10	3×10	2	11	33.5
0x3F	34.5	7	1	5×10	8×2×10	3×10	2	11	34.5

than the smallest dynamic slot in the dynamic segment. Table 5.12 shows a sample of selected dynamic frames with size and timing parameters from the test vehicle as analysed by the FAS. The frames with DLC equal to 1 are the frames with the smallest payload in the samples and have a frame duration from the TSS to CID (including DTS) of 145 bits. A frame with zero payload is also possible, however since no such frame occurred in the samples, the duration of the DTS is unknown. Since the dynamic slot occupies one or more multiples of minislots, the duration of a minislot has to be smaller than 14.5 μs plus the channel idle time for the dynamic slot. Further, two successive frames with frame ID 'n' and 'n+2' respectively will have a time difference of one minislot from the end of the dynamic slot for frame ID 'n' to the start of the dynamic slot with frame ID 'n+2'. Thereby, no frame was transmitted to occupy the minislot for frame ID 'n + 1' and the transmission of frame ID 'n+2' is then awaited until the successive minislot. The smallest duration between two dynamic frames is the time between frame ID $0x7C$ and $0x80$ from the end of the CID for frame ID $0x7C$ to the TSS of frame ID $0x80$ of 33.3 μs. The 33.3 μs are occupied by the channel idle time which also contains the minislots for the start of a frame transmission for frame ID $0x7D$, $0x7E$, and $0x7F$. Therefore the duration of a minislot is further decreased to less than 11.1 μs.

Table 5.12: Additional Dynamic Frames, Vehicle Network

Sample	Frame ID	DLC	TSS	DTS	Idle Until Next Frame	Trailer CRC passed
1	0x6C	4	7	34	115.9 μs	√
1	0x7C	4	7	34	33.3 μs	√
1	0x80	3	7	54	54 μs	√
1	0x87	4	7	34	-	√
2	0xED	4	9	34	74.2 μs	√
2	0XF7	4	9	34	1475.8 μs	√
3	0x7C	4	7	34	54 μs	√
3	0x83	4	7	34	2273.3 μs	√
4	0xD8	4	9	34	239.6 μs	√
4	0xFA	1	7	26	-	√
5	0x6E	1	7	26	46.8 μs	√
5	0x74	4	9	34	60.9 μs	√
5	0x7C	4	7	34	1044.2 μs	√
5	0x113	4	7	34	1234.9 μs	√

The duration from the last static slot to the first dynamic frame, with frame ID $0x5C$ indicates the maximum size of the static segment and the maximum amount of static slots. The minimum number of static slots was determined as 70 slots, and the time between the last static slot and the TSS of the first sampled dynamic frame indicates a maximum of additionally 21 static slots and the latest start of the dynamic segment.

No symbol was observed during the sampling and the duration of the symbol segment is therefore not specified. Further, since the dynamic segment can have minislots after the end of the dynamic slot with the highest frame ID, the start of the NIT segment is ambiguous and defined as the time after the ending of the dynamic slot with the highest recorded frame ID. The last dynamic frame in the dynamic segment in the sample data is the frame with ID $0x113$ shown in Table 5.12. The 'Idle Until Next Frame' indicates the time between the CID parameter of the dynamic slot and the TSS of the first static frame in the successive communication cycle. The maximum duration of the NIT segment, when no Symbol window is present in the network, can then be calculated using the timing information from frame ID $0x113$. The start of the NIT segment and Symbol window is then determined to occur later than 3959.15 μs after the start of the communication cycle. The earliest start of the NIT segment indicates a minimum duration for the dynamic segment. Further, when comparing the maximum duration for the minislot and the minimum duration of the dynamic segment, a minimum number of minislots can be calculated.

The results from the FAS regarding the FlexRay configuration parameters in the test vehicle are shown in Table 5.13.

Table 5.13: Estimated FlexRay Configuration Parameters, Vehicle Network

Reference	**Value**
Macrotick Cycle Length	5000 μs
Number of Static Slots (n)	$70 \leq n \leq 91$
Duration of a Static Slot	33 μs
Number of Minislots (n)	$n \geq 86$
Length of a Minislot (d)	$d < 11.1$ μs
NIT Start (t)	$t \geq 3959.15$ μs
Symbol Start (t)	$t \geq 3959.15$ μs
Payload Length in Static Segment (DLC)	8 (16 Bytes)
Number of TSS Bits (b)	$b \geq 9$ bit

5.2.3 Disconnecting a FlexRay Node

The ECU for the electromechanical power steering system was physically disconnected from the FlexRay network in order to determine the static frame transmissions originating from the ECU. Access to the vehicle's FlexRay bus was then established through the ECU connector on the BP

and BM lines, as shown in Figure 5.8.

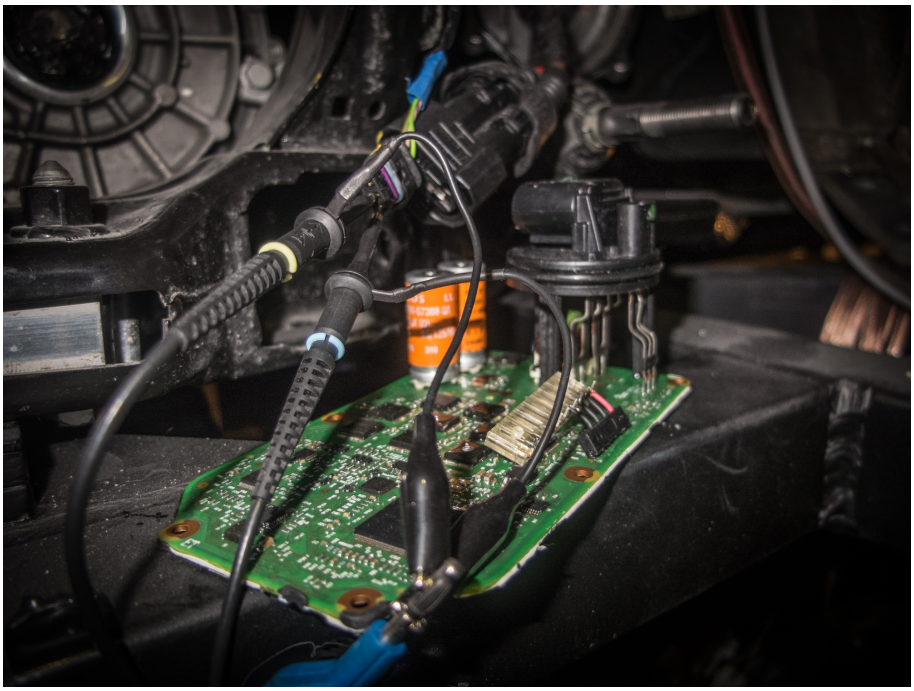

Figure 5.8: EPS Disconnected, Bus Access via Main Connector

When disconnecting the EPS ECU the communication cycle contained 21 frames instead of the original 23 as illustrated in Figure 5.9. The empty static slots are shown with an orange line. The lack of the static frame ID $0x31$ and $0x33$, when the node is disconnected, indicates that these frames originate from the ECU for the EPS system. Further, since none of the frames originating from the node has the sync frame indicator bit or the startup frame indicator bit set to a logical '1', as shown in Table 5.9, the EPS node is not a coldstart node.

Additionally, removing the node from the network caused a voltage change for the dominant bus states, increasing from 3.3 V to 3.8 V and decreasing from 1.8 V to 1.2 V for the high and low states respectively. There was also a significant increase in signal noise present in the network when the node was disconnected. The noise and voltage difference are presumably caused by the removal of the impedance-matching/termination present in the node.

5.2.4 Payload Investigation

In order to inspect the payload, multiple successive communication cycles have to be analysed in order to see how the payload changes over time. The payload alone does not give any information

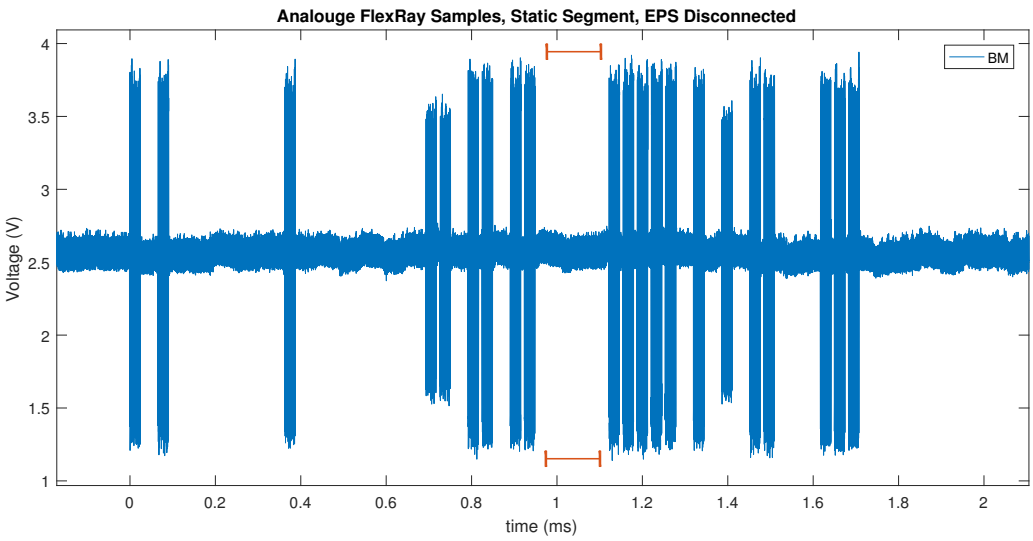

Figure 5.9: Analogue Samples, Static Frames with EPS disconnected

about the content, therefore, physical influence in the form of brake and clutch actuation are applied while sampling the FlexRay network.

A single sample containing a second of information renders the FlexRay analysis software to analyse 200 successive communication cycles. A sample rate of 5 ns with 12 bit per sample, would create a matrix of 200×10^6 rows and 3 columns per row with 12 bit samples, and would be very difficult to work with. Therefore, when analysing a longer time period, the entire sample duration is split into smaller segments and analysed individually, then the frame content is stored and compared with the successive samples.

When analysing the payload of specific frames, only the frames with the 'null frame indicator bit' set to 'one' are analysed, as the payload would otherwise only consist of 'zeros'. Another requirement for analysing the payload is that the frame 24 bit trailer CRC is equal to the trailer CRC calculated by the FAS. The 'packet sniffing' method from Section 2.7.1 is used to identify the communication between the different ECUs on the FlexRay network in the test vehicle.

Figure 5.10 shows the 16 byte payload of frame ID $0x2C$ where only the frames that passed the trailer CRC validation and had data in the payload section are plotted. 100 samples indicate samples from 100 successive communication cycles and, therefore, a time period of 500 ms. Similar plots were made for all frame IDs in all samples where the payload was analysed. The first two byte of the payload of $0x2C$ are assumed to be a two byte combination of a CRC value and

an increment counter value, consistent with the AUTOSAR end-to-end protection mentioned in 2.7.3. The same pattern are identified in many other frame IDs as a combination of byte one and two, and byte nine and 10 which implies, if the end-to-end protection is used, that it protects the six bytes succeeding the counter byte and the counter itself. Further, byte three to 10 are assumed to indicate the brake pressure, in either a combination of two bytes or expressed in one byte.

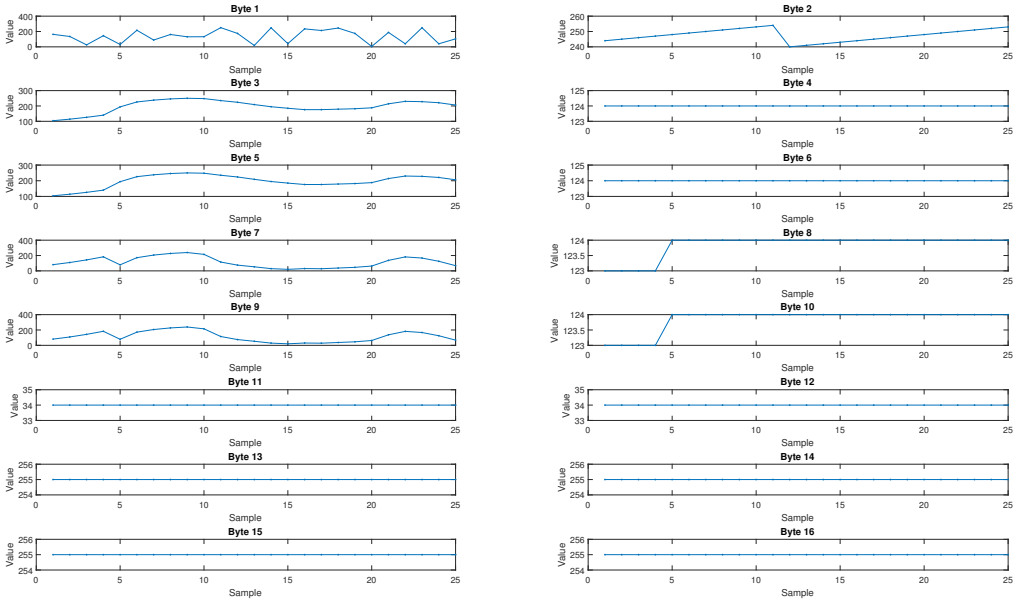

Figure 5.10: Payload Plot of Frame ID 2C, Brake Actuation

A similar behaviour when actuating the brake pedal is found in frame ID $0x2F$, except with much lower resolution. Byte 8 of $0x2F$ changes between three unique states as the brake pedal is activated, while byte 14 of the same frame changes between two states. This is indicating that byte 8 could show an active, midpoint, and standby position of the brake pedal while byte 14 shows the brake pedal activation.

As the payload in Figure 5.10 looks quite static, and does not reflect the usual behaviour of the payload in most frames, Figure 5.11 shows the payload of frame ID $0x2B$, which implements a more dynamic payload. The figure is additionally showing the possible counter and CRC combination of byte one and two, and byte nine and 10.

The frames with frame ID $0x2B$, $0x2C$, and $0x2F$ have a visible change in the payload during the brake pedal actuation which can be shown when comparing the payload to a reference

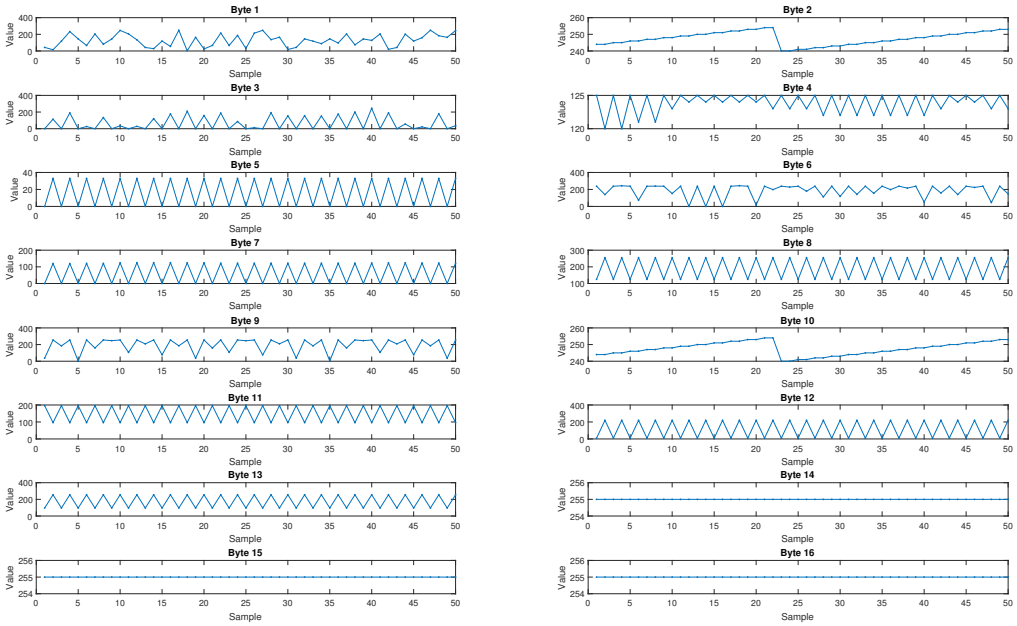

Figure 5.11: Payload Plot of Frame ID 2B, Brake actuation

payload. The reference payload is taken from a sample period with no physical interference with the vehicle (other than the natural). The payload from frame ID $0x2C$ during brake actuation (orange) plotted against the reference payload (blue) for byte one through 10 is shown in Figure 5.12. The plot are showing a fixed value for the reference payload and a dynamic pattern in the brake actuation payload. Therefore indicating that the payload of frame ID $0x2C$ is connected to the physical brake system.

A similar approach as for the brake pedal actuation was used when analysing the payload during clutch pedal actuation. During the clutch pedal activation the payload of frame ID $0x29$ and $0x3D$ responded to the actuation while the reference payload was static. The payload of frame ID $0x29$ is shown in Figure 5.13 with the clutch actuation (orange) compared to the reference payload (blue).

Additional payload analysis was done in order to investigate if an Inertial Measurement Unit (IMU) is transmitting accelerometer measurements on the network. An external force was applied to the test vehicle using the motion platform in order to create a 1 Hz, 4 cm vertical motion during the sampling of the FlexRay transmission lines. One frame ID ($0x15$) had a payload

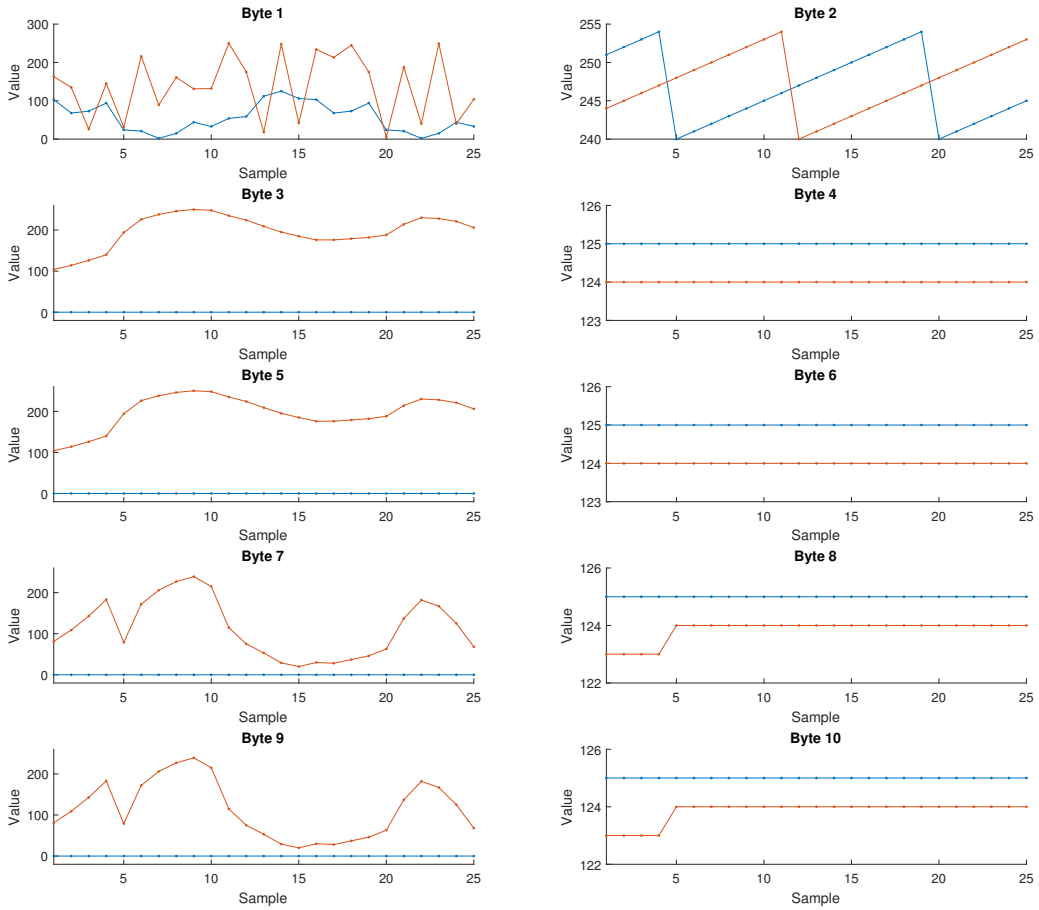

Figure 5.12: ID 0x2C, Byte 1..10, Brake Actuation Compared to Reference Payload

different to the reference payload during the testing, however, no direct link between the physical external action and the varying payload was found. The vertical motion (orange) compared to the reference payload (blue) is shown in Figure 5.14.

The results from the payload analysis for the brake actuation, clutch actuation and vertical motion are presented in Table 5.14.

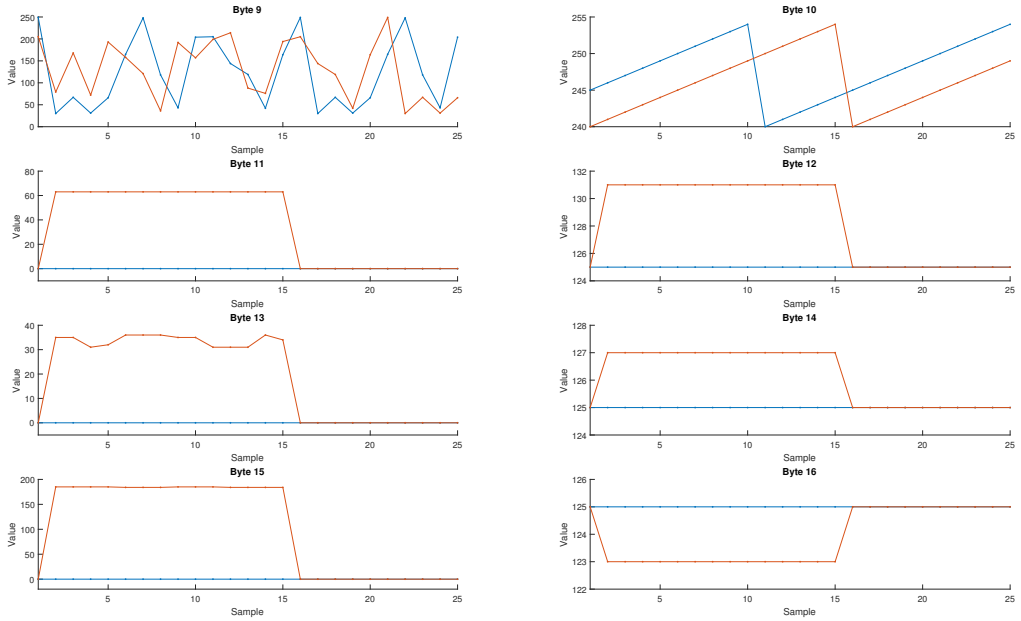

Figure 5.13: ID 0x29, Byte 9..16 Clutch Actuation Compared to Reference Payload

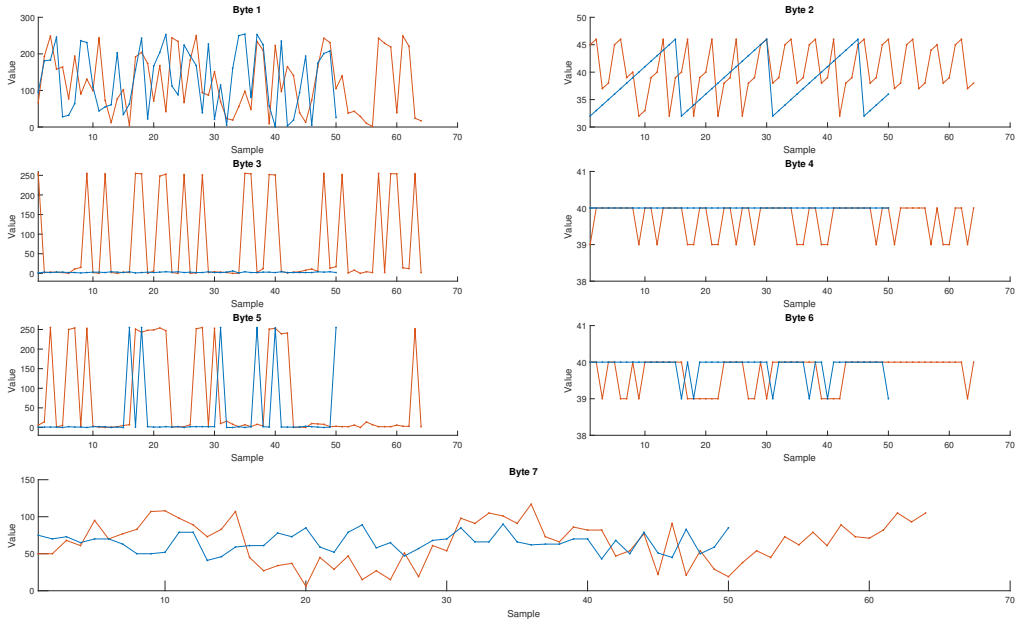

Figure 5.14: ID 0x15, Byte 1..7, Vertical Motion Compared to Reference Payload

Table 5.14: Payload Analysis, Test Vehicle

Frame ID	Explanation	Payload Byte	Hexadecimal Value
0x15	Contains Information from IMU	3,4	-
0x2B	Contains Information from Brake Pedal	3,4,6	00..FF
0x2C	Brake Pressure	3,5	00..FF
0x2C	Brake Pedal Position	7,9	00..FF
0x2C	Brake Pressure Activation	4,6	7C/7D
0x2C	Brake Pressure Situation (hard/normal/none)	8,10	7D/7C/7B
0x2F	Brake Pedal Position	8	00..07
0x2F	Brake Pedal Activation	14	F8/F9
0x29	Clutch Pedal Activation	11,12,14,15,16	00/3F, 7D/83, 7D/7F, 0/B9, 7D/7B
0x29	Clutch Pedal Activation and Position	13	0..24
0x3D	Clutch Pedal Activation	3,4,5,6	0/C0, 7D/7B, 0/C0, 7D/7B
0x3D	Clutch Pedal Activation and Position	7	0..21

Chapter 6

Suggestions for FlexRay Adversaries

Due to the deterministic aspect and the network wide configuration of FlexRay, implementing a man-in-the-middle attack on a FlexRay network requires a lot of knowledge about the original network. A simple direct connection to the transmission lines will only give access to the existing communication in the network. This is different than e.g. for CAN were a node could passively connect to an existing network and then initiate frame transmissions on the network. Further, in order to manipulate the existing communication in a FlexRay network, the frames have to be intercepted after transmission from the FlexRay node by an adversary performing a MitM attack. The manipulation could of course be conducted by reprogramming the original FlexRay nodes, as shown in Section 2.7.1, where ECU flashing was performed in the original ECUs of the vehicle, however, that is beyond the scope of this paper.

The main goal for the adversary on FlexRay is to manipulate the data content of the frames such that the receivers will process the 'adjusted' data content of the frame instead of the original. In order to process the manipulated frame, the frame has to be approved by the FlexRay communication controller of the receiver. That means that the 24 bit CRC in the trailer portion of the frame has to be calculated for the manipulated payload and transmitted with the manipulated frame.

The general idea behind an adversary on FlexRay is illustrated in Figure 6.1, where specific frames transmitted during the static segment of the communication cycle from the Node Under Attack (NUA) are subjected to an adversary. The adversary then manipulates the payload and trailer portion of the frames and transmits the frames to the remaining nodes in the FlexRay

cluster.

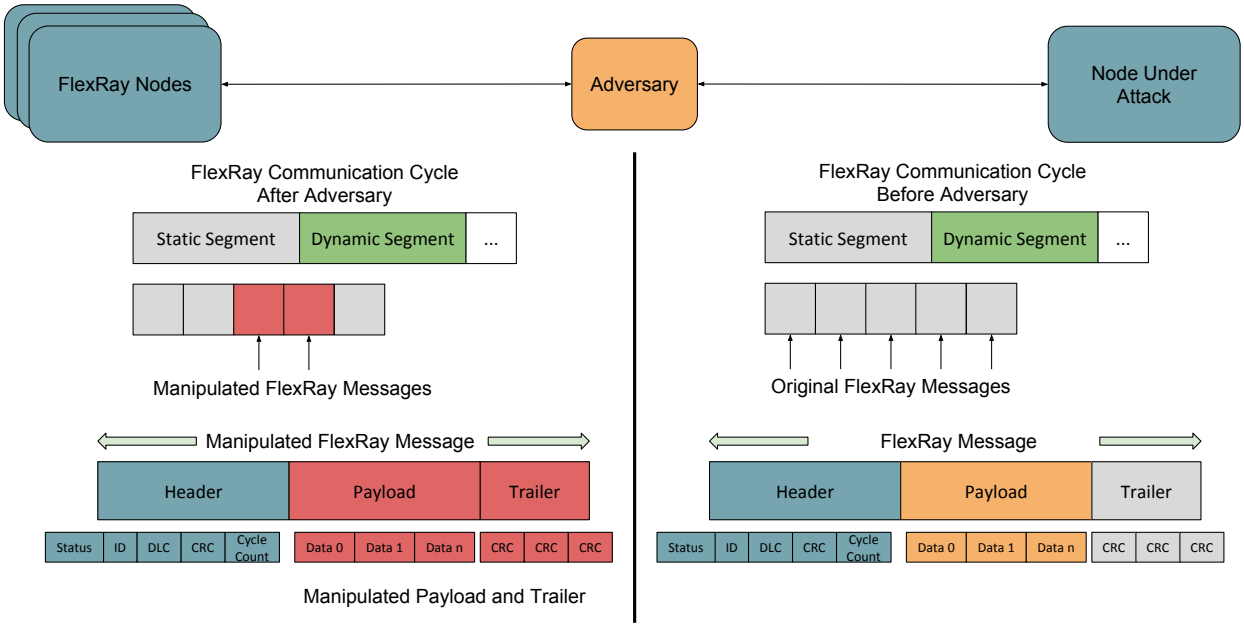

Figure 6.1: FlexRay Adversary, Manipulating Payload

The suggestions for FlexRay adversaries are theoretical and not tested out on an operating FlexRay network. Two main solutions for FlexRay adversary are suggested in this paper: 'FlexRay Adversary Implemented with Hardware Logic' and 'FlexRay Adversary Implemented with a Microcontroller', and will be explained in the following sections.

6.1 FlexRay Adversary Implemented with Hardware Logic

An adversary on FlexRay implemented with hardware logic (HWL) is suggested for manipulating the signals directly on the transmission lines while simultaneously monitoring the bus activity in order to synchronise the system with the FlexRay network. The FlexRay adversary is illustrated in Figure 6.2.

6.1.1 Monodirectional Adversary

A suggestion for a monodirectional FlexRay adversary is illustrated in Figure 6.3, where the hardware logic functions as an attacker. The adversary manipulates or passively re-transmits

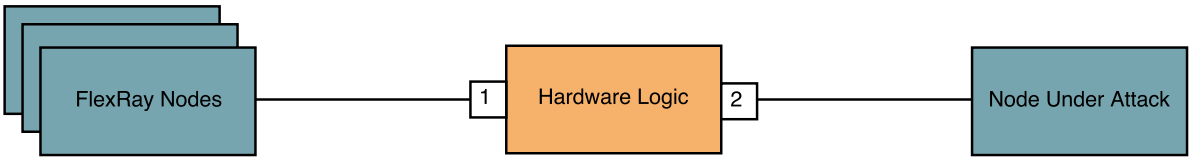

Figure 6.2: FlexRay Adversary, Hardware Logic

the data transmission from the attacking node to the rest of the nodes in the FlexRay cluster. The monodirectional FlexRay adversary consists of a data input buffer for the differential signal, hardware logic, output line driver and a bidirectional analogue switch. The hardware logic is constructed such that the FlexRay transmission and frame content can be decoded in real-time and has the capability of external data input for manipulating payload. Further, a 24 bit trailer CRC algorithm needs to be implemented in the HWL or provided by external data input. The 11 bit header CRC could be stored in a static message buffer for specific messages as the CRC only depends on the sync frame indicator, startup frame indicator, frame ID and the payload length, which does not change for a specific frame.

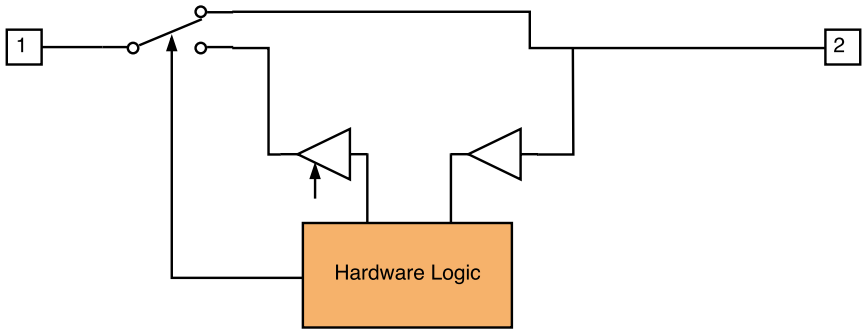

Figure 6.3: FlexRay Adversary, Monodirectional

In order to manipulate or re-transmit a FlexRay frame, the HWL needs to have the directional information of the transmitted frame. In case of a monodirectional adversary, the HWL only needs to know which frames/slots are transmitted from the NUA.

The method used in Section 5.2 for determining the origin of a FlexRay frame, where a node is physically disconnected from the network, is suitable in order to determine the origin of frames transmitted during the static segment. However, this method is not suitable for frames transmitted during the dynamic segment. The frames transmitted during the dynamic segment

are most likely application related and can be initiated as a response to dynamic or static frames or frame content. Hence, removing a node from the network might also discard frame transmission from other network participants. Therefore, a frame manipulation in the dynamic segment without the FlexRay configuration information from the network designer, is nearly impossible with the monodirectional FlexRay adversary. In order to manipulate frames in the dynamic segment, the minislot and dynamic transmissions preceding the minislots for the manipulated frame ID are monitored, such that the frame manipulation will only take place at a free minislot and with the correct priority in the dynamic segment.

6.1.2 Bidirectional Adversary

The bidirectional adversary is essentially two monodirectional adversaries for transmitting the FlexRay communication both to and from the NUA. For the implementation of the bidirectional adversary, the manipulated frames' origin needs to be determined by either side '1' or '2' of the adversary in order for the HWL to activate the corresponding input buffers and output line drivers.

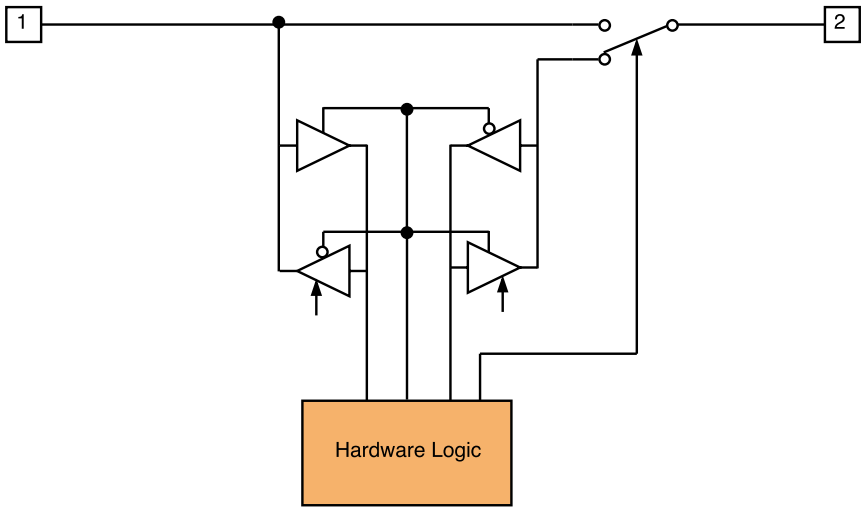

Figure 6.4: FlexRay Adversary, Bidirectional

If the entire network communication is known, and the transmission in the dynamic segment are conform to a specific pattern, the bidirectional adversary could be implemented without the bidirectional analogue switch and thereby routing all the network transmission through the

HWL. Consequently, the dynamic transmission needs to be conform to a pattern based structure, which is unlikely due to the event-triggered aspect of the dynamic frame transmissions.

For both the monodirectional and bidirectional adversary, the physical implementation might pose an additional delay through the bidirectional analogue switch, which needs to be considered as an effect for the receivers.

6.1.3 Frame Manipulation

Both 'in-frame manipulation' and 'replaced frame manipulation' are methods suggested for manipulating FlexRay frames with an FlexRay adversary implemented with hardware logic for both the monodirectional and bidirectional adversary.

6.1.3.1 In-Frame Manipulation

The in-frame manipulation method uses the bidirectional switch in order to disconnect the NUA from the FlexRay network in a specific point in global time before the frame to be manipulated is transmitted. The action point for the bidirectional analogue switch is determined by the HWL by using the timing information from the preceding frame. After switching the bidirectional switch, the HWL would be in charge of the communication between the node and the rest of the network. The HWL then intercepts the original frame transmission from the NUA and passively re-transmits the header portion while monitoring the transmission lines. Instead of continuing with the passive re-transmission, the HWL sends a different, pre-calculated, equally long and formatted bit sequence in the payload and trailer section of the frame. After the last bit of the trailer section the HWL switches back to passive re-transmission, and further switching off the bidirectional switch when the channel idle bus state is reached. Thereby the network is put back to the original operation. The in-frame manipulation of a static slot is illustrated in Figure 6.5.

The payload and the trailer portion of the specific frame are then manipulated, and the trailer must contain a valid 24 bit CRC for the data content of the entire frame, meaning that the CRC has to be calculated over the original header portion and the manipulated payload portion. Further, the status portion of the header section must have the null frame indicator bit set to 'one' in order for other nodes to process the payload of the manipulated node. The header portion is

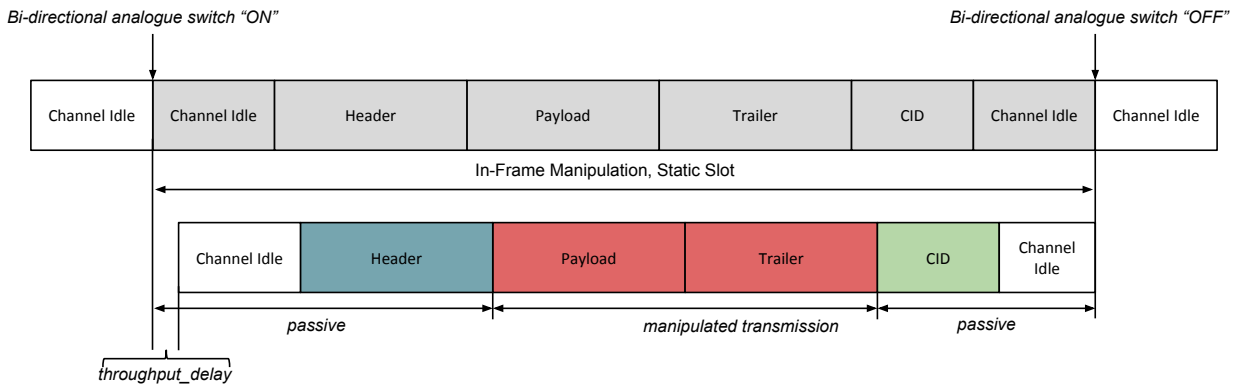

Figure 6.5: In-Frame Manipulation

then static for all information transmitted after the status portion for a specific communication cycle number (due to the iterative cycle count), giving the HWL time to calculate the trailer CRC. Further, the HWL could verify a specific configuration of the status portion, and only enable manipulation if the status portion of the current frame is equal to a pre-defined status portion.

The in-frame manipulation could also be applied such that the header portion of the frame is also manipulated. The passive re-transmission would then end upon the start of the frame.

An example is shown in Table 6.1 where payload and trailer content of frame ID $0x2C$ are manipulated. The manipulated content of the frame, byte 3 and 5, are linked to the brake pressure as shown previously during the payload analysis of the FlexRay communication in the test vehicle, Table 5.14. Given that the NUA transmits the original frame, the frame will interpreted by the receivers as brake pressure equal to zero, while in reality the brake pressure could be much higher. The manipulated frame could however be dismissed by the receivers if the payload contains an additional security measure, like AUTOSAR End-to-End protection (see Section 2.7.3). In the case where the AUTOSAR End-to-End protection is implemented in the payload, the CRC8 byte must also be manipulated. The CRC polynomial for the algorithm is unknown, however, can be deduced from the frame ID, the counter and the protected sections of the payload since these are all known. The implementation of such a protection is application specific and might be different in different systems. The trailer CRC for the manipulated frame in Table 6.1 is calculated with the same CRC algorithm used in the FlexRay analysis software and is shown in Appendix B Section 10.2.

The in-frame manipulation will introduce a transmission delay, *throughput_delay*, of the

Table 6.1: Example of FlexRay Frame Payload and Trailer Manipulation

Frame	Header	Payload	Trailer
Original	20 2C 10 E2 9F	A3 F4 68 7C 68 7C 51 7B 51 7B 22 22 FF FF FF FF	F6 B5 3A
Manipulated	20 2C 10 E2 9F	A3 F4 **00** 7C **00** 7C 51 7B 51 7B 22 22 FF FF FF FF	**02 5D 5E**

frame due to the physical signal passing through the input buffer, HWL and the output line driver. In order to ensure that the frame is received within the correct microticks the delay should be equal or less than the tolerable limitations of the asymmetric delay for a receiver as if there was an active star present in the network (see Section 2.5.9).

6.1.3.2 Replaced Frame Manipulation

The 'replaced frame manipulation' works in a similar manner as the in-frame manipulation technique. However, instead of passively re-transmitting, the HWL replaces the entire FlexRay slot. The start time of the frame that is going to be manipulated is known by using the time from preceding frame. Further, the HWL can compensate for a delay through the bidirectional switch with reducing the channel idle duration of the current frame. In this way, the manipulated frame is not subjected to any new delays due to the manipulation. The replaced frame manipulation does, however, not have the capability of responding, in the same communication cycle, to a malfunction in the NUA since the entire frame is replaced at the exact point in (global) time the original frame is transmitted.

For the replaced frame manipulation, the header, payload and trailer portion need to be configured by the HWL and match the configuration of the original frame. Since the header portion of the frame is also manipulated, the adversary has the possibility of transmitting a frame with data in the payload portion of the segment even if the original frame has the null frame indicator bit set to 'zero'. The cycle count for the replaced frame must also be iterated compared to the value of the preceding communication cycle. The replaced frame manipulation is illustrated in Figure 6.6.

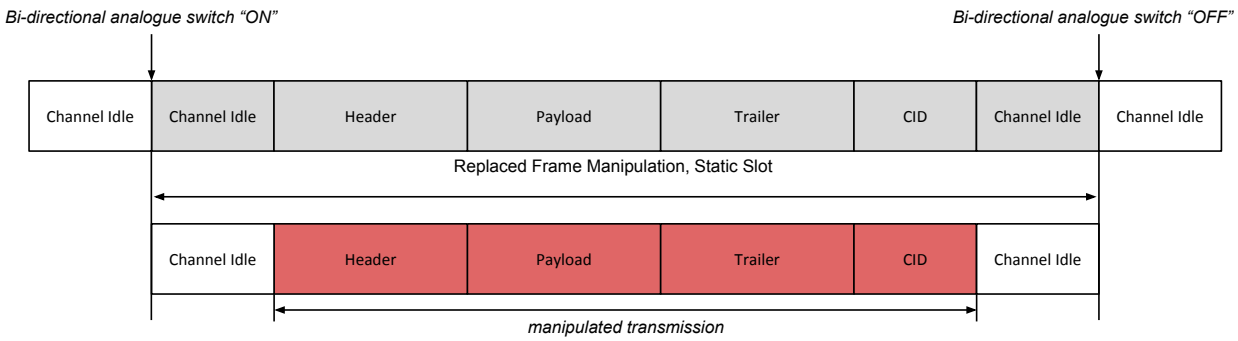

Figure 6.6: Replaced Frame Manipulation

6.2 FlexRay Adversary Implemented with a Microcontroller

A Flexray adversary solution could also be implemented on a higher level with a microcontroller solution. The NUA is then separated from the original network by an additional FlexRay node working as an active star with the extra functionality to intercept and manipulate information regarding the entire communication cycle, transmitted to and from the NUA. A solution like this is far more complex than the HWL adversary, as introducing a new operational node to an existing FlexRay network is not possible due to the global FlexRay network configuration. Therefore, the additional FlexRay node cannot interfere with the FlexRay configuration of the original network, and the communication needs to act in accordance with the same persistent global time.

As a direct re-transmission within the same bit duration is not possible due to the added processing delay, propagation delay and so forth, the adversary has to delay the entire communication cycle with at least one cycle. Consequently, the original globally synchronised network is divided into two locally synchronised networks where each of the nodes in the local networks is unaware of the communication cycle delay. The FlexRay adversary node solution is illustrated in Figure 6.7.

Table 6.2 shows the delayed communication cycle differences between the two local networks where k is the number of delayed communication cycles. The adversary FlexRay node has to decrease the cycle count value in the header section of each frame such that it will match the current communication cycle in each local synchronised network. Further, during the network startup phase, if coldstart nodes exist on different sides of the FlexRay adversary node then the entire startup phase will be delayed with k communication cycles and is conflicting with the

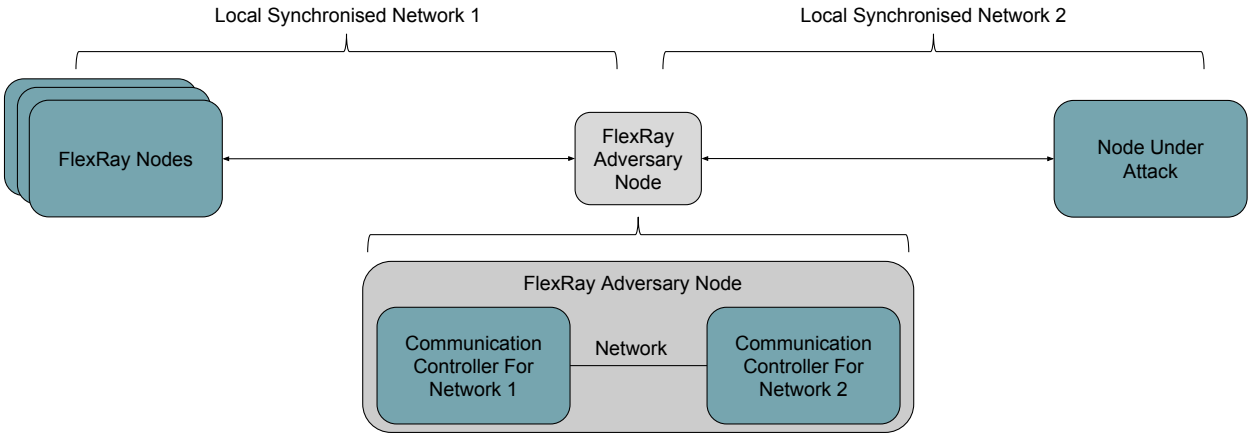

Figure 6.7: FlexRay Adversary Node

pre-defined number of communication cycles in the startup phase. The adversary would then have to falsify the sync and startup frames in the local network that occupy the leading coldstart node in order to initiate full network communication, and then, after the startup phase, send the original frames from the coldstart nodes located in the other local network with an decreased cycle count.

Table 6.2: Adversary, Communication Cycle

Communication Cycle:	Cycle Start	Cycle 2	Cycle 3	Cycle 4	Cycle n
Local Network 1:	cycle 1	cycle 2	cycle 3	cycle 4	cycle n
Local Network 2:	-	cycle k	cycle k+1	cycle k+2	cycle n-k

Since every frame in the communication cycle is passed through the bus driver, communication controller and processed by the host controller, the node has access to all the information in both header and payload segment of all frames in the network and manipulation can be performed offline in the host controller. The frame content is then loaded into an output buffer and transmitted after k communication cycles. The FlexRay adversary node would have to implement two communication controllers and bus driver systems such that each local network is connected to its own communication controller. Utilising the available transmission channel 'B' for a network that only implements communication channel 'A' would cause an inconsistency in the trailer portion of the FlexRay frames as the 24 bit CRC algorithm uses different initialisation vectors for the different communication channels. The 24 bit CRC in the trailer portion of a FlexRay is calculated online by the communication controller, meaning that the frames transmit-

ted from the original network on transmission channel ‘A’ would not be verified by the FlexRay adversary if transmission channel ‘B’ is used from the network and vise versa. The host controller then has to support two communication controllers, or two host controllers would control each communication controller independently while transmitting FlexRay network content between each other.

Chapter 7

Discussion

7.1 The FlexRay Analysis Software

The FlexRay Analysis Software (FAS) was developed while this paper was in progress and provides message content and timing information for all FlexRay messages in an analogue sample file taken from the transmission lines BP and BM of FlexRay. The FAS had some problems with handling the input data when a lot of noise was present on the transmission lines, whereas the differential value of the signal had a larger voltage threshold for the different bus states. For example, when the BP line is 4 V and the BM line is 1 V, this would produce a 3 V differential value, which is outside the threshold area for the 'High' bus state. Hence, that specific sample would be an undefined bus state. The frame containing the undefined bus state would then not pass the trailer CRC comparison because the 24 bit CRC calculated by the FAS would not be equal to the one in the trailer portion of the transmitted frame. The noise and voltage discrepancies in the FAS are, however, assumed to be fixed if the FAS adapts the voltage threshold for the bus states according to the current sample data, rather than only using the theoretical values.

Further, a solution could be implemented where only one of the transmission lines of FlexRay is sampled in order to analyse the network communication. This will come at a cost of reduced tolerance to electromagnetic interference. However, the FAS would then only need to process the analogue values from one transmission line. Therefore, both the size of the input data and the processing time of the FAS would be reduced.

7.2 Estimating the FlexRay Configuration Parameters

The FlexRay configuration parameters were estimated in Section 5.1.1 for the FlexRay test network and in Section 5.2 for the FlexRay network in the test vehicle. The estimated parameters in the test network were in some cases identical to the configured parameters in the FlexRay node, and in other cases only determined by an upper or lower value. In order to further tighten the estimates, more samples should be analysed from the network as analysing over a larger time frame might reveal additional frame transmissions in the dynamic segment. Additionally, the configuration parameters could be limited by the specifications in the FlexRay protocol specification such that an invalid value for a parameter could not be estimated (e.g maximum size of the static segment).

The estimated FlexRay configuration parameters in Chapter 6 do not include all the configuration parameters for a FlexRay node, only the ones that can be deduced by the timing information. The configuration of a FlexRay node includes additional global and local configuration parameters. Additional methods have to be used in order to estimate more parameters.

7.3 Analysing the Content in a FlexRay Network

Performing an analysis of the messages and message content is not uncomplicated in FlexRay, as the global network configuration of the time-triggered architecture renders a node unable to inject messages to the network using a simple passive bus connection. Further, the proprietary data communication and payload format can be analysed in order to prove a correlation with a physical system. However, it does not prove causation. In order to determine exactly what is transmitted in each message in the network, the FIBEX file for the specific communication network should be consulted.

In order to provoke a reaction in the message payload when investigating the payload of the messages in the test vehicle in Section 5.2.4, an external physical system in the vehicle was actuated. Although, this seemed efficient when analysing the sample file during the physical actuation, it is possible that the reaction originated from a completely different system in the vehicle. Therefore, additional testing in order to determine the physical origin of the frame content should be conducted, such as comparing the payload to external sensors and reference payloads taken under different circumstances.

7.4 A FlexRay Adversary

The use of different communication protocols and multiple networks in a vehicle offers various attack surfaces for a potential adversary. The communication of a network can be altered given that the attacker has physical access to the transmission lines of any communication protocol. For CAN, the manipulation can be done by using a passive connection to the network in order to initiate frame transmissions. Further, a MitM attack on CAN can be implemented with an additional CAN node, as the messages are not expected to arrive at a specific point in time and will therefore not be affected by the additional delay of the node.

A similar approach is not possible for time-triggered network architectures as in FlexRay due to the TDMA scheme for the bus access. Two main implementations for a FlexRay adversary are shown in Chapter 6, which takes a theoretical approach to the manipulation.

The 'FlexRay adversary implemented with hardware logic' is manipulating the frame content in real-time, directly on the transmission lines. Therefore, if there was no extra delay due to the additional circuitry of the adversary, as in case of the 'replaced frame manipulation' method, then the physical layer would be completely unaware of the adversary. The 'in-frame manipulation' method uses a passive re-transmission and manipulation solution for the frame manipulation and will, therefore, introduce an additional delay to the frame transmission. The added delay must then be sufficiently small such that it will not be noticed by the receivers which will set high demands for the hardware components used in the adversary.

An adversary implemented with a microcontroller, as mentioned in Section 6.2 is a far more complex adversary than the hardware logic adversary. For the microcontroller adversary, the attacker is implemented as an additional node, performing a MitM attack in the original network. Therefore, the node has to have all the information necessary in order to communicate on the network. Many of the FlexRay configuration and communication parameters can be deduced from the network communication as shown in Chapter 5. However, the parameters need to be identical to the configured parameters. A completely accurate parameter estimation is only guaranteed if the network is analysed during all possible situations for an infinite time, as a dynamic frame transmission is event-triggered. Consequently, implementing an additional node as a FlexRay adversary would require the configuration information of the original network

(the FIBEX file). However, if given enough time and the possibility of confirming the network transmission of a FlexRay network, the node configuration could be brute-forced using the estimated configuration parameters as a basis.

Additional research and testing have to be done in order to determine how a potential 'bus guardian' would react to the introduction of an adversary node in the network where the bus guardian is activated.

Appropriate precaution should be taken when implementing an adversary on an automotive communication network as the adversary could conflict with safety critical message transmission. Moreover, a FlexRay adversary has a large potential endangerment as the protocol is most likely used by X-by-wire applications. When manipulating the frame content of a specific frame in the communication, the original frame content would be delayed with at least one communication cycle duration, which could be valuable milliseconds during a critical situation. In the extreme case that the original frame content contains information that would lead to an immediate airbag deployment, and the same frame is manipulated in the specific communication cycle, the airbag deployment would not occur until the successive communication cycle. Further, if the frame is manipulated for every communication cycle, the airbag deployment information might never be transmitted further than to the adversary. Moreover, implementing an adversary without having proper information about the content in the network can cause serious endangerment to the driver and the surrounding environment.

The number of attack surfaces of a vehicle increase as the numbers of electronic systems, sensors and communication networks are increasing. Automotive manufacturers then have to take appropriate precautions in order to prevent unwanted access to the safety-critical information in the vehicle. The adversary solutions and frame manipulation methods mentioned in this paper are meant to raise awareness of the fact that such a manipulation is achievable while also explaining the possible consequences.

Chapter 8

Conclusion

8.1 FlexRay Analysis and FlexRay Adversary

The developed FlexRay Analysis Software (FAS) presented in this paper was tested on a FlexRay network where the communication and configuration were known in order to prove proper functionality of the program. The frame information was validated with calculating a new header CRC and trailer CRC for each specific FlexRay frame. The calculated CRCs were then compared to the respective portions in the original FlexRay frames, in which the header and trailer CRCs were identical for all frames in the sample data. Thus, the FlexRay frame content was correctly analysed by the FAS. Additionally, the frames were validated by comparing the frame format in the FAS to the FlexRay frame format, which showed a correct frame format for all frames in the sample data. In conclusion, the developed FAS was able to correctly calculate all the frame information of a given FlexRay sample file containing only the analogue sample values from the FlexRay transmission lines. Further, the timing information of the frame transmission was calculated by the FAS and provided valuable information in order to estimate some of the global FlexRay configuration parameters.

The content of the FlexRay frames was analysed over a longer time period, in which an actuation of a physical system was introduced, in order to determine the physical origin of the frame content. With the actuation of physical systems, a correlation between some of the physical systems and the payload of specific frames was found.

Multiple adversary solutions for FlexRay were suggested and thoroughly discussed and ex-

plained in order to provide necessary information for practical implementations. A FlexRay adversary was proven to be possible for either the manipulation of specific frames or the entire network communication. Therefore, implementing a man-in-the-middle attack on FlexRay is possible.

8.2 Suggestions for Further Work

A suggestion for further work is to implement automatic FlexRay parameter identification in the FAS in a similar manner to what is accomplished in the paper *"Automatic Parameter Identification in FlexRay based Automotive Communication Networks"* by [1]. Further, the FAS could be implemented as a real-time system such that the 'packet sniffing' over time would be a much easier process and could be compared in real-time. In order to investigate the frame content (payload) in an automotive network more easily, the FAS could process an automatic payload identification whereas the physical actuation is monitored with external sensors and an automatic correlation algorithm could then identify the physical systems in the payload of the FlexRay frames.

In order to make the FAS more tolerant to signal noise, automatic voltage threshold adjustment should be implemented. Additionally, if a 'timing error' has occurred, whereas one or multiple portions of a specific FlexRay frame are short by one or more multiple bit values, the FAS could "guess" a bit value, and calculate and compare the new trailer CRC such that the timing error could be corrected.

The adversary suggestions presented in this paper have to be proven in a physical system in order to show how the adversaries would work in a FlexRay network. Therefore, further work regarding the FlexRay adversaries could deal with the implementation of both the hardware logic and the microcontroller adversary in a FlexRay network performing a man-in-the-middle attack on a specific node in the network cluster.

Chapter 9

Appendix A

9.1 Acronyms

μT Microtick

ABS Anti-Lock Braking System

AP Action Point

BM Bus Minus (Transmission line)

BP Bus Plus (Transmission line)

BSS Byte Start Sequence

CAN Controller Area Network

CAS Collision Avoidance Symbol

CID Channel Idle Delimiter

CRC Cyclic Redundancy Check

DLC Data Length Coding

DTS Dynamic Trailing Sequence

ECU Electronic Control Unit

EPS Electromechanical Power Steering

ESC Electronic Stability Control

FAS FlexRay Analysis Software (Developed for this paper)

FES Frame End Sequence

FIBEX Field Bus Exchange Format

FSS Frame Start Sequence

FTDMA Flexible Time Division Multiple Access

GTDMA Global Time Division Multiple Access

HWL Hardware Logic (for Adversary)

ID Identifier

LIN Local Interconnected Network

MCU Microcontroller Unit

MitM Man-in-the-Middle

MT Macrotick

MOST Media Oriented Systems Transport

NIT Network Idle Time

NUA Node Under Attack

TDMA Time Division Multiple Access

TSS Transmission Start Sequence

V_Bus Voltage difference between the BP and BM Bus lines

Chapter 10

Appendix B

10.1 MATLAB Implementation of 11 Bit Header CRC

This section contains the FlexRay 11 bit header cyclic redundancy check used by the FlexRay Analysis Software.

```
% 11 Bit Header CRC Algorithm for FlexRay Analysis Software
function CRC = Header_CRC(Message_Header)
%Initialise Variables
header = strcat(Message_Header(1,:),Message_Header(2,:)...
    ,Message_Header(3,:));
header = uint32(hex2dec(header));
header = bitshift(header,-1);
header = bitand(header,hex2dec('FFFFF'));
CrcInit = int32(hex2dec('1A'));
data_length = int32(20);
CrcNext = int32(0);
CrcPoly = uint64(hex2dec('385'));
CrcReg_X = uint64(CrcInit);
header_temp = uint64(0);
reg_temp = uint64(0);
header = bitshift(header,11);
CrcReg_X = bitshift(CrcReg_X,21);
```

```
CrcPoly = bitshift(CrcPoly,21);
% CRC Algorithm
while(data_length)
    data_length = data_length -1;
    header = bitshift(header,1);
    header_temp = uint64(bitand(header,hex2dec('80000000')));
    reg_temp = bitand(CrcReg_X,hex2dec('80000000'));
    if (bitxor(header_temp,reg_temp))
        CrcNext = 1;
    else
        CrcNext = 0;
    end
    CrcReg_X = bitshift(CrcReg_X,1);
    if(CrcNext)
        CrcReg_X = bitxor(CrcReg_X,CrcPoly);
    end
end
% Format Output
CrcReg_X = bitshift(CrcReg_X,-21);
CrcReg_X = bitand(CrcReg_X,hex2dec('00000000000007FF'));
CRC = dec2hex(CrcReg_X);
end
```

10.2 MATLAB Implementation of 24 Bit Trailer CRC

This section contains the FlexRay 24 bit trailer cyclic redundancy check used by the FlexRay Analysis Software.

```
% 24 Bit Trailer CRC Algorithm for FlexRay Analysis Software
function CRC = Trailer_CRC(Message_Header,Message_Payload,Channel)
% Format data input
Message_Header = dec2bin(hex2dec(Message_Header),8);
```

```
Message_Payload = dec2bin(hex2dec(Message_Payload),8);
temp_header = '';
for i= 1:size(Message_Header,1)
    temp_header = strcat(temp_header,Message_Header(i,:));
end
temp_payload = '';
for i= 1:size(Message_Payload,1)
    temp_payload = strcat(temp_payload,Message_Payload(i,:));
end
% Initialise Variables
data_long = strcat(temp_header,temp_payload);
if strcmp('B',Channel)
    % Initialisation Vector Channel B
    CrcInit = int32(hex2dec('ABCDEF'));
else
    % Initialisation Vector Channel A
    CrcInit = int32(hex2dec('FEDCBA'));
end
CrcNext = int32(0);
CrcPoly = uint64(hex2dec('5D6DCB'));
CrcReg_X = uint64(CrcInit);
header_temp = uint64(0);
reg_temp = uint64(0);
CrcReg_X = bitshift(CrcReg_X,8);
CrcPoly = bitshift(CrcPoly,8);
data = uint32(bin2dec(data_long(1:31)));
data_length = size(data_long,2);
bit_length = size(data_long,2);
i = 1;
% CRC Algorithm
while(data_length)
    data_length = data_length -1;
```

```
37      data = bitshift(data,1);
        if i <= bit_length-31
39      data = bitor(data,bin2dec(data_long(i+31)));
        end
41      i = i+1;
        header_temp = uint64(bitand(data,hex2dec('80000000')));
43      reg_temp = bitand(CrcReg_X,hex2dec('80000000'));
        if (bitxor(header_temp,reg_temp))
45          CrcNext = 1;
        else
47          CrcNext = 0;
        end
49      CrcReg_X = bitshift(CrcReg_X,1);
        if(CrcNext)
51          CrcReg_X = bitxor(CrcReg_X,CrcPoly);
        end
53  end
    % Format Output
55  CrcReg_X = bitshift(CrcReg_X,-8);
    CrcReg_X = bitand(CrcReg_X,hex2dec('0000000000FFFFFF'));
57  CRC = dec2hex(CrcReg_X);
    end
```

10.3 MATLAB Implementation of FlexRay Analysis Software

The MATLAB implementation of the FlexRay Analysis Software are shown in the following link: https://drive.google.com/drive/folders/0B4duzcMTjDZbNVhoUy1KNks0aDA?usp=sharing

Bibliography

[1] Armengaud, E., Steininger, A., and Horauer, M. (2006). Automatic parameter identification in flexray based automotive communication networks. *Vienna University of Technology and University of Applied Sciences Technikum Wien*.

[2] Bosch, R. (1991). Can specification 2.0 part a and part b. Technical report, Robert Bosch GmbH.

[3] Consortium, F. (2005a). Flexray protocol specification version 2.1 rev. a. Technical report, FlexRay Consortium.

[4] Consortium, F. (2005b). Flexray requirements specification version 2.1. Technical report, FlexRay Consortium.

[5] Delgrossi, L. and Zhang, T. (2012). *Vehicle Safety Communications - Protocols, Security, and Privacy*. Wiley, Hoboken, New Jersey.

[6] Furst, S. (2011). Autosar and functional safety. In *Safetronic 2011*, Sheraton Arabellapark Hotel, Munich. BMW Group.

[7] Huse, M. I. (2016). On-board communication systems in vehicles - flexray. *Norwegian University of Science and Technology - Department of Engineering Cybernetics*.

[8] Instruments, N. (2016). *White paper: FlexRay Automotive Communication Bus Overview*. `http://www.ni.com/white-paper/3352/en/`.

[9] Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., and Savage, S. (2010). Experimental security analysis of a modern automobile. *University of Washington and University of California San Diego*.

[10] Miller, C. and Valasek, C. (2015). *Remote Exploitation of an Unaltered Passenger Vehicle*. `http://illmatics.com/Remote%20Car%20Hacking.pdf`.

[11] Paret, D. (2007). *Multiplexed Networks for Embedded Systems, CAN, LIN, Flexray, Safe-by-Wire...* Wiley, West Sussex, United Kingdom.

[12] Paret, D. (2012). *Flexray and its Applications, Real Time Multiplexed Network*. Wiley, West Sussex, United Kingdom.

[13] Randy, F. (2004). *X-By-Wire: For Power, X Marks the Spot*. `http://electronicdesign.com/automotive/x-wire-power-x-marks-spot`.

[14] Sikora, A., Berbineau, M., Vinel, A., Jonsson, M., Pirovano, A., and Aguado, M. (2014). *Communication Technologies for Vehicles*. Springer, Springer Cham Heidelberg New York Dordrecht London.

[15] Smith, C. (2016). *THE CAR HACKER'S HANDBOOK*. William Pollock, No Starch Press, Inc. 245 8th Street, San Francisco, CA 94103.

[16] Vector, I. G. (2016). *XL Driver Library manual*.

[17] Wolf, M., Weimerskirch, A., and Paar, C. (2004). Security in automotive bus systems. *escrypt*.