DEGREE PROJECT IN ELECTRICAL ENGINEERING, SECOND LEVEL
STOCKHOLM, SWEDEN 2019

Enabling Database-based Unified Diagnostic Service over Local Interconnect Network

Tian Xu

KTH ROYAL INSTITUTE OF TECHNOLOGY
Electrical Engineering and Computer Science

Abstract

Unified Diagnostic Service (UDS), which is an international and not a company-specific standard, is used in almost all new electronic control units (ECUs) by now. Modern vehicles have a diagnostic interface for off-board diagnostics, which makes it possible to connect a diagnostic tool to the vehicle's bus system like Controller Area Network (CAN) and Local Interconnect Network (LIN). However, as the most commonly used method, sequential method on the UDS data transmission over LIN does not only result in low reliability and flexibility but also fails to meet the standard for LIN development defined in the latest LIN specification published by the consortium. With standard workflow and application interfaces, this Master Thesis will develop and evaluate a database-based method to build a UDS system over LIN, where all the information for the network is defined in the LIN database, and the protocol properties are realized in a reusable model so that it can be easily reconfigured for the future development of other services.

As a result, a new method including a layered-structure LIN protocol model and a LIN database has been successfully designed and implemented. The prototype is built on the device PIC32MX795, and the database can be deployed by the configuration tool to specify the UDS communication schedule. Further, several performance evaluations have been performed. The tests indicate that the system is qualified on the limited hardware platform and the configuration flexibility is proved by different databases.

Keywords

LIN (Local Interconnect Network), UDS (Unified Diagnostic Service), ECU (Electronic Control Unit), Database, PIC32MX795

Abstrakt

Unified Diagnostic Service (UDS), som är en internationell och inte en företagsspecifik standard, används nu i nästan alla nya elektroniska styrenheter (ECU). Moderna fordon har ett diagnostiskt gränssnitt för diagnostik utanför kortet, vilket gör det möjligt att ansluta ett diagnostiskt verktyg till fordonets bussystem som Controller Area Network (CAN) och Local Interconnect Network (LIN). Som den mest använda metoden resulterar emellertid sekventiell metod på UDS-dataöverföringen via LIN inte bara i låg tillförlitlighet och flexibilitet utan uppfyller också standarden för LIN-utveckling som definieras i den senaste LIN-specifikationen publicerad av konsortiet. Med standard arbetsflöde och applikationsgränssnitt kommer denna masteruppsats att utveckla och utvärdera en databas-baserad metod för att bygga ett UDS-system över LIN, där all information för nätverket definieras i LIN-databasen, och protokollegenskaperna realiseras i en återanvändbar modell så att den enkelt kan konfigureras för framtida utveckling av andra tjänster.

Som ett resultat har en ny metod som inkluderar en LIN-protokollmodell med skiktstruktur och en LIN-databas framgångsrikt designats och implementerats. Prototypen är byggd på enheten PIC32MX795, och databasen kan konfigureras av verktyget för att ange UDS-kommunikationsschema. Vidare har flera prestationsutvärderingar genomförts. Testen indikerar att systemet är kvalificerat på den begränsade hårdvaruplattformen och konfigurationsflexibiliteten bevisas av olika databaser.

Nyckelord

LIN (Local Interconnect Network), UDS (Unified Diagnostic Service), ECU (Electronic Control Unit), Databas, PIC32MX795

Acknowledgments

I would like to express my sincere appreciation to my Master thesis examiner Professor Zhonghai Lu and supervisor Dr. Yuan Yao for the irreplaceable help and suggestion on both technological and academic parts of my work.

I would like to thank my industrial supervisor Attila Fodor from BlueAir Cabin AB for giving me this interesting topic and sufficient support during the whole project. I am very grateful for the time spent in the company with all the other excellent colleagues as well as Thomas from ihr Germany, for the patient suggestions and help.

Finally, super thanks to my parents, my friends for supporting me throughout the whole master's program. Big thanks to my girl-friend, Yiwen Jin for the understanding and concern. Wish you the best PhD career in UK.

Stockholm, September 2019

Tian Xu

Table of contents

List of figures

List of tables

List of acronyms and abbreviations

LIN	Local Interconnect Network
OEM	Original Equipment Manufacturer
ISO	International Standardization Organization
OSI	Open System Interconnection
UART	Universal Asynchronous Receiver/Transmitter
ECU	Electronic Control Unit
UDS	Unified Diagnostic Services
CAN	Controller Area Network
AUTOSAR	Automotive Open System Architecture
MISRA-C	Motor Industry Software Reliability Association for C
API	Application Programming Interface
PDU	Packet Data Unit
PID	Protected Identity
NAD	Node Address
PCI	Protocol Control Information
SID	Service Identity
LDF	LIN Description File
PDUR	Protocol Data Unit Router
DCM	Diagnostic Communication Manager

1 Introduction

This chapter describes the project's background and defines the problem, purpose, and main goals of the project. An outline of the project's structure is given at the end of the chapter.

1.1 Background

Founded by five automakers (BMW, Volkswagen Group, Audi Group, Volvo Cars, Mercedes-Benz), Local Interconnect Network (LIN) is a concept for low-cost automotive networks, which complements the existing portfolio of automotive multiplex networks. LIN is widely used in the data transmission between components in vehicles for further quality enhancement and cost reduction purposes. In recent years, many car manufacturers, tool developers, and component suppliers have embedded Unified Diagnostic Services (UDS) in the LIN as a standard for the management of vehicle diagnostic functions within both future applications and standard software models, which provides unified interfaces for diagnostic services in electronic control unit (ECU). Over the last few years, there has been a tremendous increase in the functional scope of automotive electronics and the applications of software in vehicles. The complexity of the vehicle electronic systems is also increasing with the growing software quantity. While the UDS standard unified the diagnostic application standard, it is also necessary to improve the flexibility of automotive product modification to reduce the development resource, eliminating repetitive low-level development work. The latest LIN specification provides a database-based method to define the network, including the LIN nodes, signals, frames, and the network's behavior. The database describes the network properties from an abstract level and can be helpful for the communication system design. At present, the database generation has been set as the first step in the standard workflow, followed by the design of the LIN protocol model.

UDS is a diagnostic communication protocol in the electronic control unit environment within the automotive electronics, which is defined in the ISO 14229 standard [1]. Before the foundation of UDS, the vehicle's maintenance can only rely on the experience of the workers, since the information of the fault is not easy to acquire. UDS can provide efficiency and convenience by providing a unified diagnostic interface and a series of specific services. Once the components are out of order, they will save the fault information in the memory, and the maintenance worker can read the information through the communication bus. For example, when an ECU experiences an under-voltage fault, it will store the diagnostic trouble code, and optionally saves snapshot information when the fault occurs (such as the current speed of the vehicle). Currently, UDS is supported as a standard for the diagnostic and calibration of vehicle parameters by automotive open system architecture (AUTOSAR). The protocol is used to diagnose errors and reprogram ECUs, combining and consolidates standards such as ISO 14230 or Diagnostics on CAN (ISO 15765), and it's independent of vehicle manufacturers. UDS is based on the Open Systems Interconnection (OSI) basic reference model in accordance with ISO 7498-1 and ISO/IEC 10731, shown in Table 1.

Applicability	OSI seven layers	Enhanced diagnostics services				
Seven layers according to ISO/IEC 7498-1 and ISO/IEC 10731	Application (layer 7)	ISO 14229-1				
	Presentation (layer 6)	Vehicle manufacturer specific				
	Session (layer 5)	ISO 14229-2				
	Transport (layer 4)	ISO 15765-2	ISO 10681-2	ISO 13400-2	Not applicable	ISO 17987-2
	Network					

	(layer 3)					
	Data link (layer 2)	ISO 11898-1, ISO 11898-2	ISO 17458-2	ISO 13400-3, IEEE 802.3	ISO 14230-2	ISO 17987-3
	Physical (layer 1)		ISO 17458-4		ISO 14230-1	ISO 17987-4

Table1: UDS layer model

The UDS in the LIN network focuses on 3 of the 7 layers of the OSI model: physical layer, data link layer, and application layer. A detailed description of the area should be moved to Chapter 2, where more information about the background is given together with the related work.

The economic efficiency of LIN is partly due to the feasibility of utilizing UART for building a LIN cluster, running unconditional data transmission and diagnostics information. When calling the LIN tasks for UDS, the cluster will be predefined and the transmitting byte data will be generated manually to fill each section of a standard LIN frame. However, the existing sequential method provides little flexibility when there is a need to apply the network to another application, which requires the developers to rebuild and configure the whole LIN framework. As a result, it is helpful to find out if there is a reusable and reliable method for LIN communication and how it works on different UDS services and protocol versions.

1.2 Research problems

The thesis aims to merge the reusable and flexible LIN protocol stack model with a specific UDS use case defined in the database. The implementation process suggests the following research problems:

- How to compose the LIN database to describe the LIN communication?

- How to design the layered structure of the LIN protocol model?
- How to apply the LIN APIs defined in the specification in the LIN model implementation?
- Compared with the sequential method, will this new model be capable of LIN communication?

1.3 Purpose

The purpose of this thesis is to design and implement a database-based method to realize the UDS on LIN. From the users' perspective, the transmitting signal selections can be switched by changing the configuration bit in the database, without making complex modification in the packaged LIN protocol model. The LIN model that provides interfaces to different applications and hardware platforms will bring great convenience to other LIN services in future development.

1.4 Goals

The goals of this thesis are to build a high-level database that defines UDS functions in a LIN cluster and implement a packaged LIN protocol model including the inner data transmission and the interfaces for the highest application layer and lowest hardware layer. The goals have been divided into the following five sub-goals:

- Compose a LIN Description File (LDF) database to describe the UDS in the LIN cluster in the format of master tasks and slave tasks. Basic timing constraints and LIN nodes' capability are also set in the database.
- Generate a cluster description and LIN signal description from the database as an interpretation of the network.
- Design the LIN model with interfaces to the application layer (defined in the database) and hardware layer (configuration bits of ECU).

- Build the LIN protocol model with hardware abstraction layer, LIN driver layer, LIN interface layer, and LIN communication services layer to provide a stable inner data transmission mechanism with standard LIN APIs.

- Analyze the performance of the method. Test the feasibility of the designed LIN model and compare the results with the existing sequential method.

The sequential method of LIN communication in the project indicates building the LIN master task containing a series of signal transmission operations to compose a LIN frame on the bus. The sequential method combines the application software with a hardware description, resulting in low flexibility to meet the rising variety of customer's demand. The new method in the goals will make the hardware and software independent with each other, which provides a possibility to reuse the software on different applications and ECUs, and enhance the quality of software and efficiency of development. The changes between the two methods is shown in Figure 1.

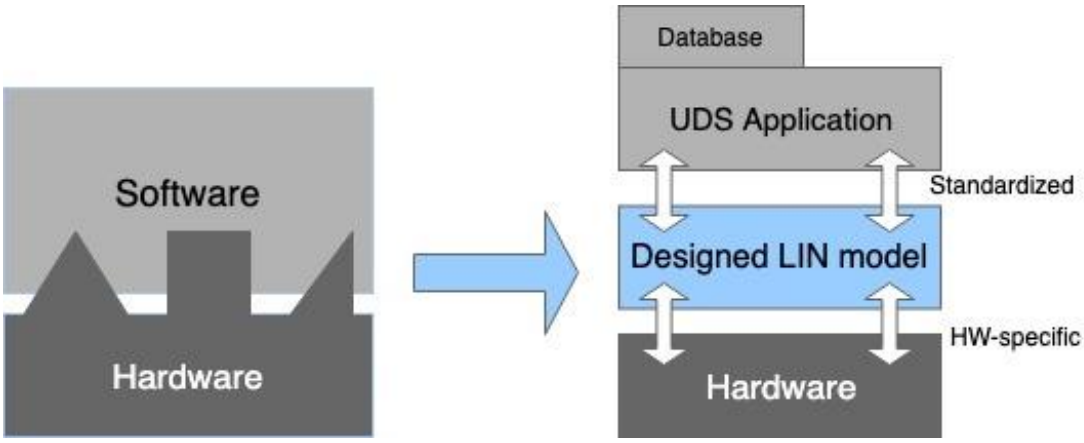

Figure 1: The change of UDS system development

1.5 Research methodology

The literature review is the first and most important step before the work. The literature review is a requirement for every robust research with qualitative or

quantitative study approach because it is the review of related literature that offers a solid theoretical foundation for the research. The quantitative research approach is used in this work to have a more accurate conclusion since the project will be verified by a variety of experimental data. To test the performance of the generated LIN system, the experimental method is used to collect and analyze the data from a series of test cases, which manipulate the UDS variables, and the comparison between methods is helpful for the evaluation and future improvement of the system. Additional details concerning methodology are given in Chapter 3.

1.6 Delimitations

This project requires some specific hardware modules, so the hardware configuration settings are only applicable to the selected platforms, but can be easily modified to be adapted to other devices. All the tests and results are generated from the selected devices, which cannot be replaced during the thesis. The UDS functions realized in the thesis will match the supported UDS schedules which are specified in the ECU's handbook. The basic functions library in C language are provided by the hardware manufacturer, Microchip, and the version of LIN specification defined in the database is version 2.1, enabling byte arrays and diagnostic APIs.

Due to the hardware platform limitations, the database configuration process will be performed on a PC instead of Linux-based hardware in the initial plan, which decreases the integrity of the network. This redirection shifts the emphasis of the project from building a database parser to the implementation of the LIN protocol model.

It is recommended in several related works to use assembly language for a more efficient execution. Due to the software limitation of the integrated development environment for MCU, C language is used instead to make the development process reliable, and the development can match the LIN API requirements defined in the LIN specification.

According to the AUTOSAR standard, only the communication service branch is selected as the target structure in the project, which can provide sufficient supports for UDS.

1.7 Ethics and sustainability

This master thesis aimed to improve the efficiency of the LIN network development by a packaged LIN model. The information of the commercial product from Microchip company is public inside the official documentation. The reference ISO standard is authorized. The ECU in the LIN cluster is exclusive in Blueair Cabin AB, and confidential in terms of code and hardware structure, it will be regarded as a black box in the thesis with the functional interface visible to users and developers.

The proposed design will provide high flexibility of the LIN model between the database describing the behaviors of the network and the hardware for both the master node and the slave node. As a result, the Blueair Cabin AB is able to reproduce the ECU or redesign the UDS without modifying the complex middle layers at the expense of lower consumption of timing and economic resource.

1.8 Outline

Chapter 2 presents the relevant theoretical background information about LIN and UDS. The brief concept and introduction used in the thesis are stated to help users understand the properties of the LIN protocol and the features of UDS. Besides, this chapter provides several related works and a discussion on them.

Chapter 3 describes the research methodology and testing devices. The entire research process is described first. The chapter also gives information on the literature study, experiment design, and data collection together with some analysis methods.

Chapter 4 introduces the implementation process of the LIN model. The idea of the layered structure design is introduced first. Then the specific implementation process of each layer is presented, which is the key point of the thesis.

Chapter 5 shows the obtained results and the analysis based on them. Chapter 6 gives a conclusion of the thesis, including some reflections and possible future work. The appendix is available in the end.

2 Theoretic background

This chapter provides a basic background related to this thesis. Section 2.1 describes the basic knowledge of the LIN network, including the network topology and concept of each node between which the data transmission occurred. The properties and utilities of UDS on LIN is presented in section 2.2, focusing on the data structure and transport layer. Several related works are given in section 2.3 with discussion.

2.1 Local interconnect network (LIN)

LIN is a serial communication protocol which efficiently supports the control of mechatronics nodes in distributed automotive applications, used for communication between components in vehicles [1]. Since more and more electronic control units are utilized in a vehicle's bus system, the requirements for a cheap serial network arose as the technologies and the facilities implemented in the car grew, while the CAN bus was too expensive to implement for every component in the car [1]. European manufacturers started using different serial communication technologies.

Figure 2: LIN consortium logo

The protocol is generated from the Volcano-Lite technology developed by the Volvo spin-out company Volcano Communication Technology (VCT). Since other car corporations were also interested in a more cost-effective alternative to CAN, the LIN syndicate was created. In the middle of 1999, the first LIN protocol (1.0) was released by this syndicate. The protocol got two updates in

the year of 2000. In November 2002, LIN 1.3 was released with updates of signals format and as well as some new features like diagnostics. The changes were mainly aimed at simplifying the use of off-the-shelves slave nodes. Examples of areas where LIN is and can be used in a car: mirrors, wiper, window lift, and rain sensors.

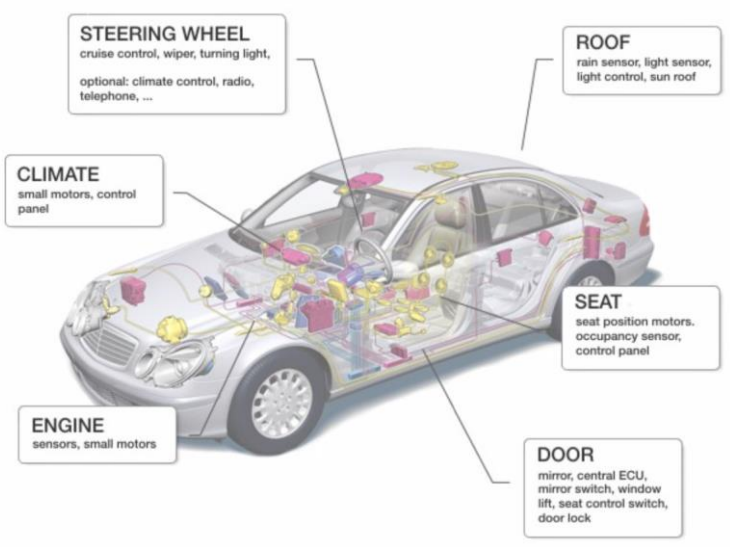

Figure 3: Areas of use of LIN in a vehicle [2]

The main properties of the LIN bus are [1]:

- Single master with multiple slaves concept
- Low cost silicon implementation based on common UART/SCI interface hardware.
- Self-synchronization without quartz or ceramic resonator in the slave nodes.
- Deterministic signal transmission with signal propagation time computable in advance
- Low cost single-wire implementation

- Speed up to 20 Kbit/s
- Signal based application interaction
- Predictable behavior
- Transport layer and diagnostic support

LIN provides a cost-efficient bus communication where the bandwidth and versatility of CAN are not required. Figure 4 shows the transmit speed and economic cost of LIN and other automotive bus protocols.

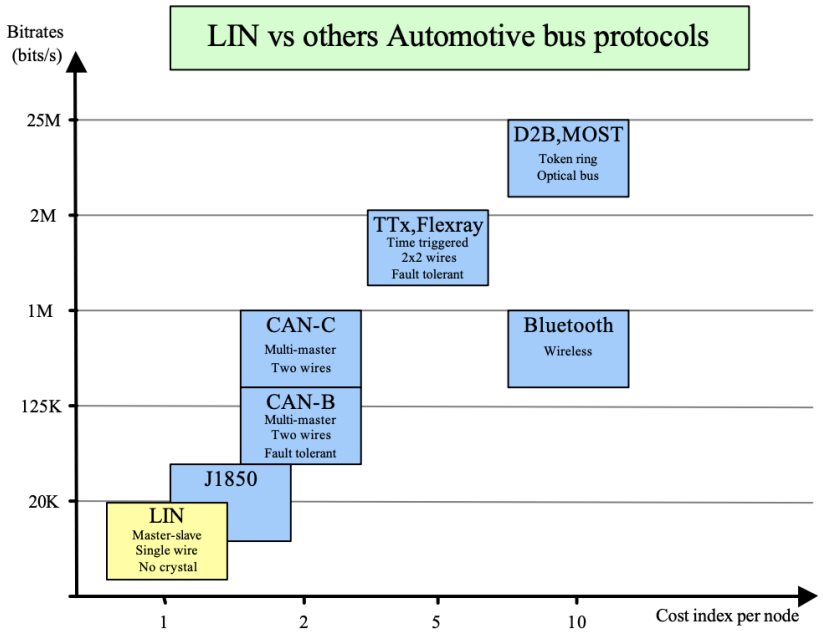

Figure 4: Speed comparison between LIN and other protocols

As Figure 4 shows, the protocol supports up to 20 Kbits/s transmission rate. LIN is self-synced by the master node and can detect errors using 8 bits checksum, combined with a classic checksum or an enhanced checksum, and 2 parity bits. Drivers that run with LIN are easy to use, non-complex, and also available in the market [14]. These drivers require less harness usage and are

more reliable for the mid-range modules. The LIN network simplicities facilitate the development of extensions for functionalities on the components.

2.1.1 Network topology

Usually, in an automotive application, the LIN bus is connected between smart sensors or actuators and an electronic control unit which is often a gateway with CAN bus.

A LIN network consists of one LIN master and up to 16 LIN slaves, every additional node will lower the network resistance by approximately 3% and threaten the stability. A LIN system with more than 16 nodes should be regarded as an unstable system.

In a LIN cluster, a master node contains a master task as well as a slave task. All other slave nodes contain only one slave task. It means a master node can act as both publisher and subscriber while a slave master node can only act as a subscriber. A typical cluster with one master node and two slave nodes is depicted below:

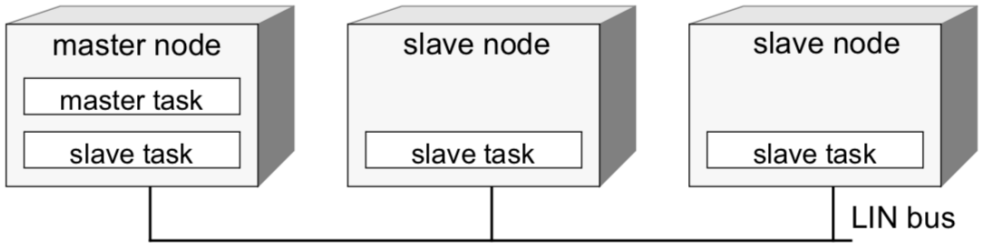

Figure 5: LIN cluster [1]

Data exchanged on the LIN bus complies to a specific frame structure. A master node contains several frames for transferring in the master task. When and which frame shall be transferred on the bus are decided by the master task. The slave tasks provide the data field transported by each frame. The ability of a frame handler is defined in the node interface layer.

2.1.2 Node concept

A node in the cluster interfaces to the physical bus wire using a frame transceiver. The frames cannot be accessed directly by the top-level application, and a signal-based interaction layer is added between. For some special functional frames, a transport layer exists between the top and the frame handler as a complement. Figure 6 shows the structure of a typical node.

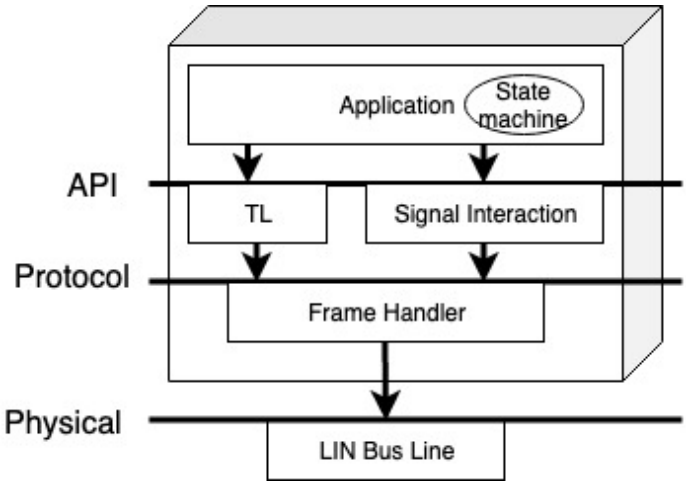

Figure 6: LIN node structure [1]

The application layer is usually accomplished by a certain application developer who wants to define a macroscopic function of the LIN cluster as well as building the node task's state machine to ensure the security of the data transmission. Going through the predefined APIs, which have already been set in the specification of LIN syndicate, the data will be packaged into packet data unit (PDU) or assigned to signal interaction by write and read operations into each section of a frame. The diagnostic frames will transfer through the transport layer to rebuild the frame structure with a protocol control information byte, and the unconditional frame aiming at data exchange will access the signal interaction component directly. Then the frame will be emitted to the physical payer, which is the LIN bus.

2.1.3 Data transmission protocol

Two types of data can be transported on the bus: signals or diagnostic messages. Signals are scalar values or byte arrays and diagnostic messages are set to meet some predefined functions number. Both kinds of data should be packed into the data field of a frame, which is a specialized structure in the LIN protocol.

A frame is made up of a header provided by the master task and a response provided by a slave task [1]. The header consists of a break field and sync field followed by a frame identifier. The frame identifier uniquely defines the functions of the frame. The slave task appointed for providing the response associated with the frame identifier transmits it, as depicted in Figure 6. The response consists of a data field together with a checksum field.

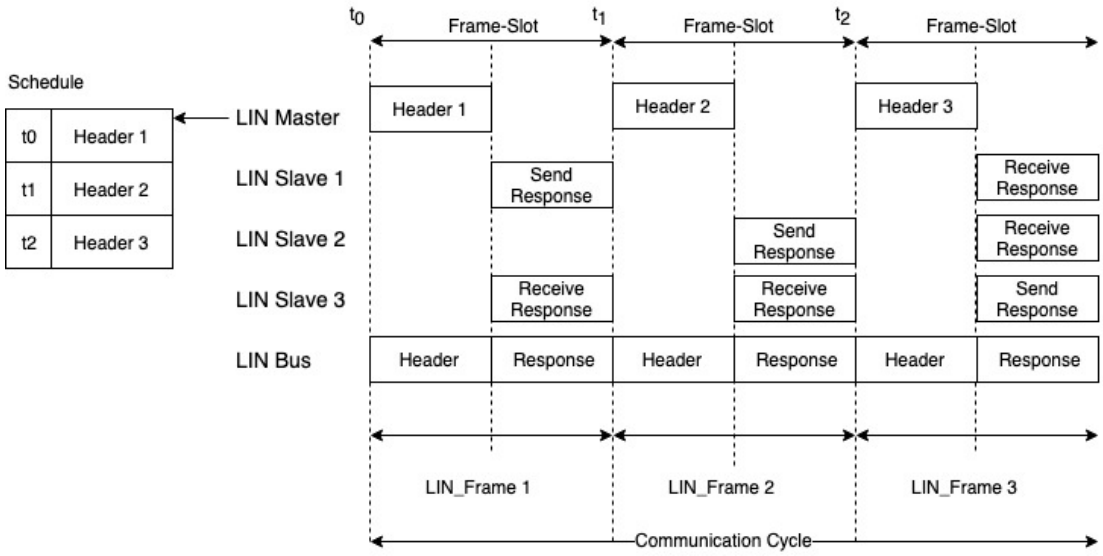

Figure 7: Node communication between master and slave nodes

As Figure 7 shows, a header sent from the master task will be responded by a subscribed slave task. All the signals compose a standard LIN frame which can be detected from the LIN bus perspective. The order and time slot for headers' transmitting are based on a predefined schedule, thus makes the communication process predictable.

A key feature of the LIN protocol is that the frame transmission is based on the schedule tables. Schedule tables make it possible to assure that the bus will never be overloaded [1]. The schedule defines the deterministic behavior of the communication tasks on the master node, and then controls the entire LIN cluster which is initiated by the master task.

The active schedule table will be processed till another requested schedule table is due. When the current schedule table's frame index reaches its end, the schedule will be started again at the beginning point of the schedule. The effective switching signal to the new schedule will be valid at the start of a frame slot. This means that a schedule table switching request will not interrupt an ongoing transmission on the bus, but should be handled in multiple sub-tables system.

2.1.4 Protocol

The entities that are transferred on the LIN bus are frames, which can ensure the publisher can be response by the correct subscriber.

2.1.4.1 Frame structure

The structure of a frame is shown in Figure 8. The frame consists of various fields, one break field followed by four to eleven byte fields, labeled as in the Figure 8.

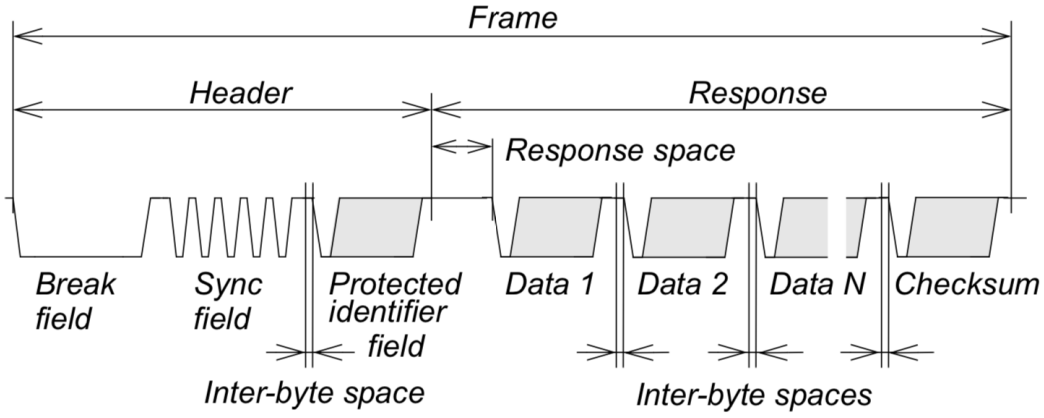

Figure 8: The structure of a frame [1]

The header starts at the falling edge of the break field and ends after the end of the stop bit of the protected identifier. The response starts at the end of the stop bit of the protected identity (PID) field and ends after the stop bit of the checksum field.

Each byte of a LIN message (except synch break) is sent according to the UART standard, see Figure 9. The least significant bit of the data is sent first and the most significant bit last. The start bit is encoded as a bit with value zero (dominant) and the stop bit is encoded as a bit with value one (recessive).

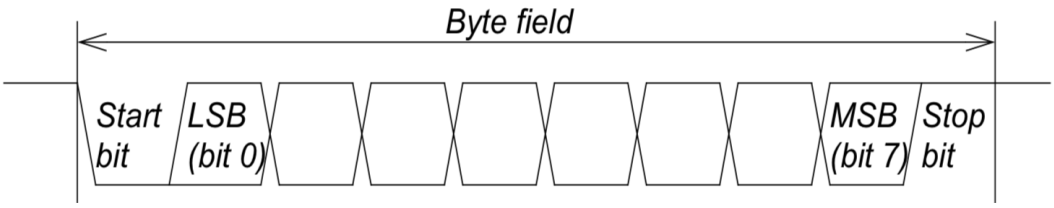

Figure 9: Structure of a byte field [1]

2.1.4.2 Frame header

The frame header, which can be regarded as the master request on the LIN bus sent by the master task, is made up of a synchronization break, a synchronization field, and a protected identifier field.

The break field is used to signal the beginning of a new frame [1]. It is the only part that does not need to comply with the data structure depicted in Figure 8. The field is generated by at least 13-bit times of dominant value followed by a break delimiter, as shown in Figure 10.

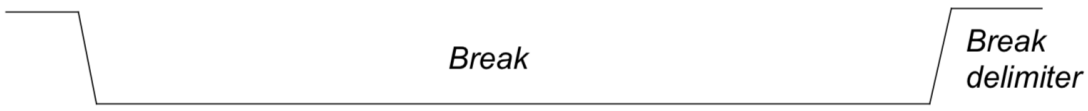

Figure 10: The synchronization break field [1]

The synchronization field provides the time slot for the slave node to regulate the baud rate for the preparation of the future transmission. Sync field is a byte field with the data value 0x55, as shown in Figure 11. When a break/sync field sequence happens, the communication tasks in progress will be aborted and processing of the new frame shall commence.

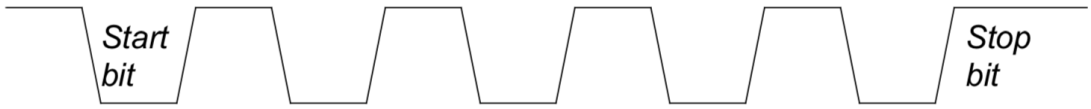

Figure 11: The sync byte field [1]

The protected identifier field consists of two sub-fields: the frame identifier and the parity, depicted in Figure 12.

Figure 12: The PID field [1]

The identification bits, ID0-ID5, indicate which message is requested from the master. It is possible to use 64 different identifiers in a LIN bus system. These are divided into three different categories, which are shown in Table 2. Only one slave can respond to a specific request and this is determined by what ID master requests.

ID	Description
0-59	Signal carrying frames
60 and 61	Diagnostic and Configuration frames
62 and 63	Reserved frames for future enhancements

Table 2: Frames with available IDs

The second part of the identifier field, P0 and P1, are two parity bits calculated according to equation (1) and (2).

$$P0 = ID0 \oplus ID1 \oplus ID2 \oplus ID4 \qquad (1)$$

$$P1 = \neg\,(ID1 \oplus ID3 \oplus ID4 \oplus ID5) \qquad (2)$$

2.1.4.3 Response

As the other part of a frame, a response is provided by a slave task that subscribes various frame headers. The response consists of a data field and a checksum field.

Following the frame header, the data field is the response part of the frame, transmitted by a specific slave node subscribing to a certain protected identifier. A frame carries between one and eight bytes of data. The number of data contained in a frame with a specific frame identifier shall be agreed by the publisher and all subscribers [1]. A data byte is transmitted as part of a byte field, which is shown in Figure 9.

For data entities that are longer than one byte, the entity least significant bit (LSB) is contained in the byte sent first and the entity most significant bit (MSB) in the byte sent last (little-endian). The data fields can hold up to eight bytes, shown in Figure 13.

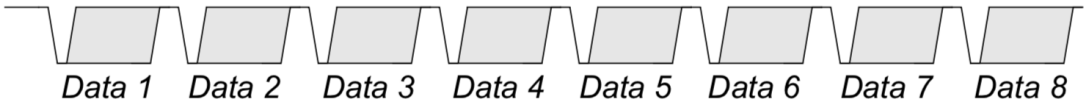

Figure 13: Data bytes field in a frame [1]

Every frame ends with a checksum to ensure the correctness of the signal transmission in the LIN cluster. The checksum contains the inverted eight-bit sum with carry bits over all data bytes or all data bytes and the protected identifier, which can be classified into classic checksum and enhanced checksum, which are respectively regarded as a classic checksum for protocol LIN 1.x and enhanced checksum for protocol LIN 2.x. Diagnostic frames with identifiers 60 (0x3C) to 61 (0x3D) shall always use classic checksum. Table 3 describes the checksum calculation of four bytes (0x4A, 0x55, 0x93, 0xE5).

The resulting sum is 0x19. The calculated sum is inverted and sent as a checksum 0xE6. The recipient then can easily check the consistency of the received frame by using the same addition mechanism. If these both correspond to 0xFF, the exchanges are correctly received.

Action	Hex	Carry	D7	D6	D5	D4	D3	D2	D1	D0
0x4A	0x4A		0	1	0	0	1	0	1	0
+0x55	0x9F	0	1	0	0	1	1	1	1	1
(+carry)	0x9F		1	0	0	1	1	1	1	1
+0x93	0x132	1	0	0	1	1	0	0	1	0
(+carry)	0x33		0	0	1	1	0	0	1	1
+0xE5	0x118	1	0	0	0	1	1	0	0	0
(+carry)	0x19		0	0	0	1	1	0	0	1
Invert	0xE6		1	1	1	0	0	1	1	0
Check	0xFF		1	1	1	1	1	1	1	1

Table 3: Example of checksum calculation [1]

2.1.5 Physical layer

To enable a connection to the microcontroller's UART, a transceiver (transmitter/ receiver) is used in each node of the network. The transceiver's main task is to convert the LIN bus's 12V signal to the UART's 5V receive/ transmit signals. The transceiver is connected to the battery positive pole (VBAT) via a termination resistor and a diode. The diode must ensure that the bus does not power the microcontroller in the event of a local power failure.

Two logical states can be sent over the LIN bus, 0 and 1, respectively. The 0 is represented by a voltage level near the ground where a limit value of 40% of the supply voltage is acceptable and the 1st of a voltage level near the supply voltage (limit value of 60% of the supply voltage), shown in Figure 14.

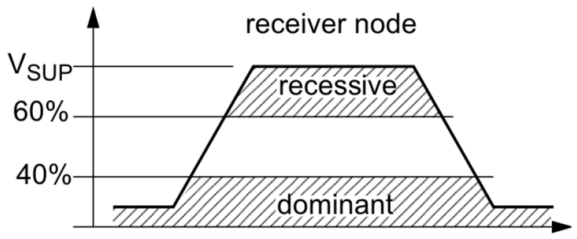

Figure 14: Voltage Levels on the LIN bus line [1]

2.2 Unified diagnostic services (UDS)

The UDS on LIN describes the method to accomplish the process for the data request and response between a client and the LIN slave nodes through or on the LIN master node. UDSonLIN references ISO 14229-1 and ISO 14229-2 and specifies implementation requirements of the following [14]:

- Diagnostic services to be used for diagnostic communication over LIN.
- Server memory programming for in-vehicle LIN servers with and external test equipment.
- Configuration of a LIN slave node as specified in ISO 17987.

Architectural, diagnostic communication performance and transport protocol are accommodated by dividing diagnostics services functionality into three diagnostic classes [14].

1. Diagnostic class I: Smart and simple devices such as sensors and actuators which are in little need of amount of diagnostic functionality. Data reading from a sensor, actuator handling and fault memory solving is completed by the master node, with the method of the signal carrying frames. Therefore, complex specific diagnostic support for the tasks is not required.
2. Diagnostic class II: Compared to diagnostic class I, diagnostic class II provides node identification support besides class I. The extended node

identification is normally required by vehicle manufacturers. Backbone testers or master nodes use ISO 14229-1 diagnostics services to send the requests of the extended node identification information.

3. Diagnostic class III: Slave nodes are devices with several additional application functions typically including local data and information processing (e.g. function controllers, local sensor/actuator loops). Therefore, the slave nodes need extended diagnostic support. The services of a control signal and raw data transmission are not exchanged within the master node and, as a result, not included in signal carrying frames. The primary difference between diagnostic class II and diagnostic class III is the distribution of diagnostic capabilities between the LIN master node and the LIN slave node for diagnostic class II while for a diagnostic class III LIN slave node, no diagnostic application features of the LIN slave node are implemented in the LIN master node [17].

According to the OSI model in Table 1, the application layer services which are defined in ISO 14229-1 are used for the client-server-based systems to perform the functionalities like the test, inspection, diagnostic monitoring or programming of on-board vehicle servers.

The message buffer is controlled by the session layer and requested by the data link layer when a message start is detected and as soon as the message length is available.

The transport layer part makes use of the network layer services which are defined in ISO 14229-2 for the transmission and reception of diagnostic messages. The units that are transported in a transport layer frame are PDU formatted. A certain message may contain one or more PDUs, which can be divided into single frame transmission and multiple frame transmission, which is made up of a first frame and several following consecutive frames according to the length byte in the message. The payload consists of node address (NAD), protocol control information (PCI), service ID (SID), length

(LEN) and request data. The PDU types which are supported in the UDS are shown in Figure 15.

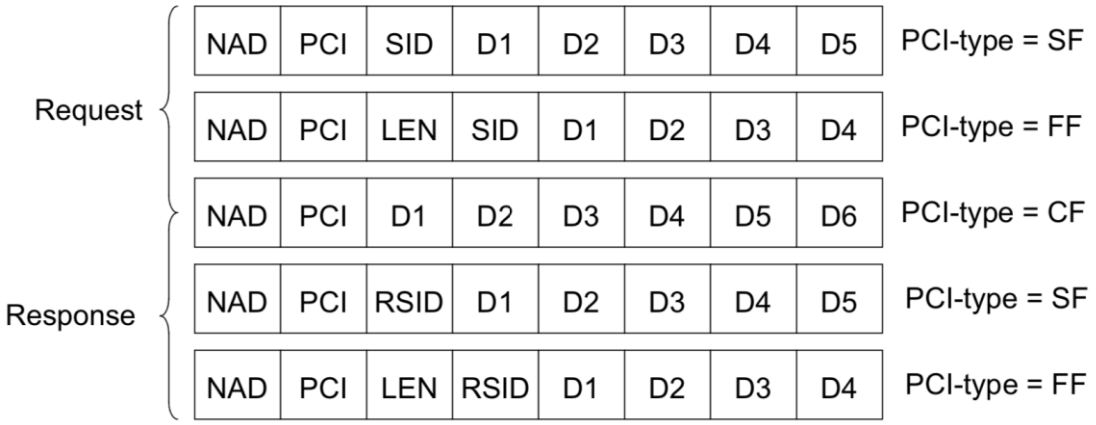

Figure 15: PDU types supported by LIN transport layer

Figure 16 shows a typical process of a timing sequence chart of a UDSonLIN through a back-bone CAN bus, within a certain timing constraint defined in the node description file [1].

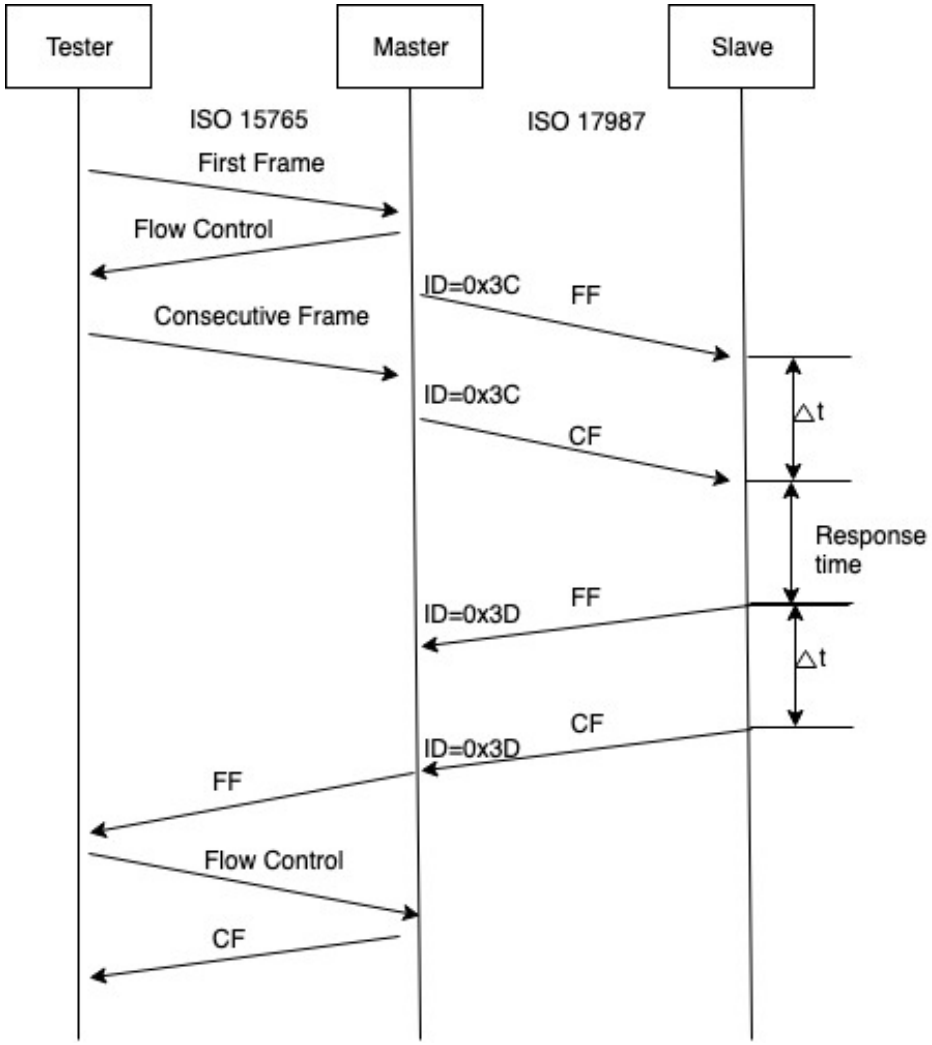

Figure 16: Timing sequence chart of a certain UDSonLIN

It is the master task's responsibility to determine when and which request frames will be emitted. Timing properties and transmission order are included in the control schedule. The master node shall receive all diagnostic requests addressed to the slave nodes from the backbone bus or master node and route them to the appropriate LIN clusters. All requests and response messages addressed to the slave nodes can be routed in the network layer.

Error handling for single frame PDU mostly happens on the exceeding amount of carrying data, thus the receiver will ignore the units with a length value greater than six bytes. For the first frame in multiple frames transmission, the receiver will ignore the message with a length value greater

than the maximum available receive buffer size of the slave node. After the reception of a single or first frame, the current process will be aborted if the NAD is not equal to the functional NAD. Also, the message reception shall be aborted because of timeout errors.

The SID field is defined by the corresponding ECU manufacturer. Table 4 describes the typical use of several SIDs, and the read by identifier services are used in the scope of this project. The SID numbering is consistent with ISO 15763-3 [14] and places node configuration 0xB0 to 0xB7 in an area defined by the vehicle of ECU manufacturer.

SID	Service
0xB0	Assign NAD
0xB1	Assign frame identifier
0xB2	Read by Identifier
0xB3	Conditional Change NAD
0xB4	Data Dump
0xB5	Reserved
0xB6	Save Configuration
0xB7	Assign frame identifier range

Table 4: Supported node configuration and identification services

If the service in Table 3 is supported in the slave node, the response will be mandatory even if the request is broadcast. The response frame shall include a specified response service identifier (RSID), which is calculated according to equation 3.

$$RSID = SID + 0x40 \quad (3)$$

Although diagnostics, node configuration, and node identification services share the same frame IDs, i.e. 0x3C for master request frame and 0x3D for slave response frame, different services identities are used for configuration and diagnostics. Node configuration services can be performed by the master task independently while diagnostic services are mostly routed on request

from test equipment from external of internal of the bus. Both cases use the same NAD and transport protocol with the exception that node configuration services are always performed within a single frame (SF). Only slave nodes have NADs, which are also regarded as the source addresses in a diagnostic slave response frame.

2.3 Related work

While the LIN protocol is open, most of its implementations are commercial products. To the best of my knowledge, the sequential LIN transmission is the most commonly used way to realize the LIN communication at present, since LIN is always used in the dedicated function module in vehicles. A recent work of a design of the CO2 measurement module based on LIN [2] by China Shipbuilding Industry Corporation applies a sequential-signals transmitting strategy to implement the UDS in LIN communication. Merging the application layers downward to hardware abstract layers, it utilizes an integrated sensor system as a probe to measure the concentration of CO2 by a unified diagnose services through LIN bus [2]. As the specific functions of the model are predefined in the work, the sequential transmitting can ensure the correctness within a limitation of services. For example, Shao's master thesis [4] implements a window control system embedded in the vehicle which cannot support signal reception service. The work is made of assembly code to gain a high efficiency, and the main function of the system is limited in control signals' sending. The sequential method cannot clearly separate master tasks and slave tasks since the transmission occurs unconditionally, and the slave tasks are triggered by a certain signal reception. Thomas and Magnus proposed a database-based method [3] to describe the properties and functionalities of the network. The LIN driver in his work uses interrupt service routines to distinguish and accomplish the signal transmission of the master task and slave task, thus strengthen the reliability of signal transmission and reception. As defined in the newest LIN specification, a standard LIN development process should follow the workflow in the specification, and utilize the APIs to realize the definition of functional scope

in the documentation. The works in [5], [6], and [7] implement the LIN system with the standard workflow. Shu provides a LIN workflow in her work [8] and [9] based on an automated LIN configuration tool, to generate the network from the database. The LDF database defines the application, and the LIN APIs separate the transmission, reception, initialization, and error detection functions, provide flexibility to modify the behavior in the nodes. With the rising complexity of automotive applications in vehicles, there is an increasing demand for reusable and flexible solutions to realize LIN communication. Luo and Xie built a LIN flash bootloader based on UDS in their project [10]. Applying standard LIN APIs, the work adopted a packaged LIN protocol model with interfaces to UDS application and MCU hardware layer. Inside the LIN driver, UART driver and timer driver provides hardware-related configuration for signal transmission, Frame handler is built to handle frames of different types, and the frames will be allocated to signal interaction and transport layer according to their types. The model is built to be reusable by dividing the application and hardware layers. The structure is shown in Figure 17.

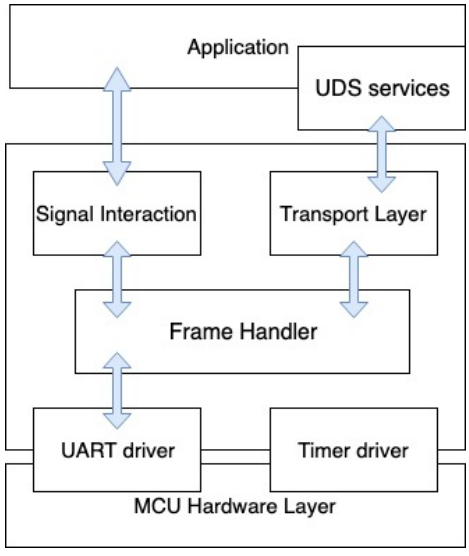

Figure 17: LIN model structure in related work [10]

Tan et al. apply a similar LIN protocol model in their work [11]. The LIN APIs used in the project are provided by the MCC library instead of LIN standard APIs, which is not valid at present.

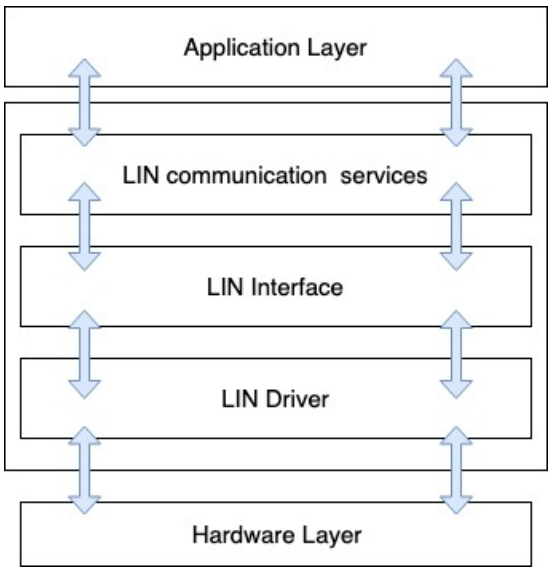

Figure 18: LIN model structure in related work [12]

To strengthen the flexibility of automotive products, the AUTOSAR standard is developed by several companies. More detailed information about AUTOSAR is given in Chapter 4. In a recent MSc project by Li [12], an AUTOSAR-based LIN model structure was defined, and all the LIN services like state management, diagnostic and data transmission are runnable in the model by the layered framework. The work is generated from the basic software layers of AUTOSAR, shown in Figure 18. Compared with the work of Luo and Xie, this LIN model is relatively integrated. The implementation of the project [13] builds a similar AUTOSAR-based LIN protocol model. The function of the work is to acquire diagnostic information within a LIN cluster, running on PC. The methods in these works can significantly change the relationship between software development and hardware development, and the development of the standardized software components can be reused for other applications and ECUs. Our work differs in two ways from the projects above. Firstly, the LIN protocol model is implemented on MCU with LIN APIs

defined in the LIN specification instead of a PC to build an independent function module instead of a testing module. Secondly, the behavior and physical features of LIN communication are defined in the database above the application layer to provide better integrity. Since the in-between layers can be well packaged, the services can be changed for the high level without taking protocol properties modification into consideration.

3 Methodologies and methods

The purpose of the chapter is to describe an overview of the research methodologies used in this project. Section 3.1 describes the specific research process. Section 3.2 shows the selected hardware platform and software development tools. The method used for data analysis is included in section 3.3. Section 3.3 introduces some testing tools used in the verification stage.

3.1 Research process

The classical research methods are used in the thesis, shown in Figure 19. The research process started with a use case of data acquisition from a Blueair Cabin ECU in a predefined LIN cluster, and the initial research problem was to find the solutions of the UDS system implementation. The literature review stage includes reviews on related websites, dissertations, journals and the datasheet of selected hardware platforms. The work is completed in the lab of Blueair Cabin AB, thus the handbooks of the company's previous products are another source of reference. After the literature review, the research problem has been clarified and more specific. Also, the knowledge about rules and specifications of used protocols and techniques in the project have been acquired, and the results of related works have a great reference value, shown in section 2.3.

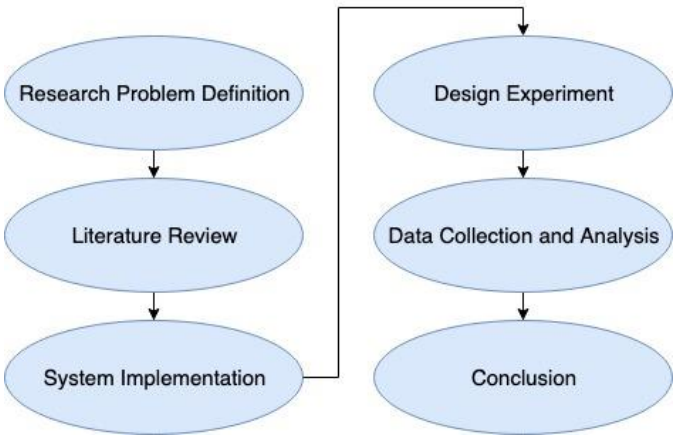

Figure 19: Research process

In the system implementation stage, the connection of the hardware system and the development of software applications have been made. The MCU is loaded with the database and the designed LIN model, providing interfaces to the application and the hardware connection. The connection between MCU and LIN transceiver is based on the UART port and wired connection. The signal of transmission and reception will process an adaption operation between MCU (LIN master node) and the ECU (LIN slave node) by the transceiver on physical layers.

The software implementation of the work follows a standard waterfall software developing process shown in Figure 20. The requirements are collected from the literature review and research problem, and the requirements focus on the potential users who will use the LIN model and the UDS system.

The software structure design is the most important during the design and implementation, and the idea of the structure in this thesis comes from a series of related works and related standards. The verification and maintenance are continuously in need to maintain the reliability of the software, also the potential LIN services will be taken into consideration in future development.

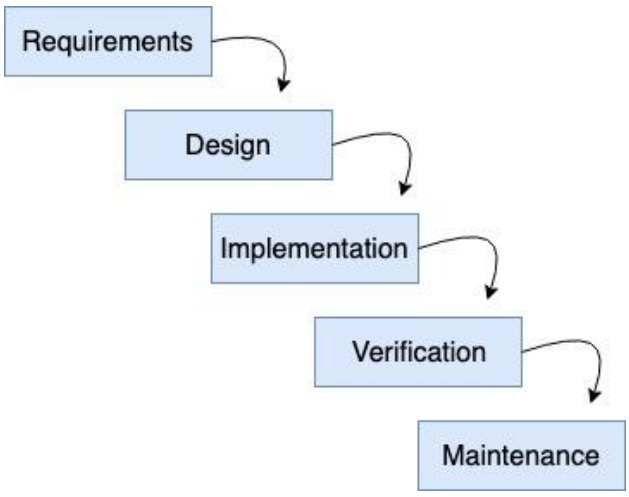

Figure 20: Software layer implementation process

A performance evaluation of different methods of LIN communication will be done in the test stage. The existing sequential method is used for making the comparison of the performance of two methods in the experimental method. Most of the evaluations will focus on the designed LIN model which is reusable to various usage of the LDF database, as well as the feasibility of the UDS communication through the LIN network. The test cases are generated from the handbook of the target ECU. Some non-statistics results will be proved with different databases, which will be tested by the oscilloscope, shown in section 3.3.

In the data collection and analysis section, some parameters will be collected in the local buffer to indicate the feasibility of the tested UDS service and database. The data will be collected to analyze the compilation performance of the LIN model. After the collection, the Excel will be used to analyze and plot several figures to give a more intuitive view. The results will be used to make a final summary and conclusion.

3.2 Hardware and software selection

This section describes the hardware platform (section 3.2.1), software (3.2.2), and target device chosen for testing in the project (section 3.3.3).

3.2.1 Hardware selection

The project will be performed on an MCU from PIC32 series. The main features of the selected MCU PICMX795F512L (Figure 21) are shown in Table 5 below, and the UART port is used to build the LIN communication with a LIN transceiver and a pull-up circuit. The MCU will act as a LIN master including the scheduling of master tasks and slave tasks. The physical features and configuration bits of the hardware are defined in the database and abstraction layer. The project is runnable on other derivatives of the same MCU series since they are equipped with the same UART system.

MCU Core	80MHz/105DMIPS, 32-bit MIPS M4K Core

	2 x CAN2.0b modules with 1024 buffers
	32x32-bit Core Registers
	Fast context switch and interrupt response
System Features	512K Flash
	128K RAM
	Programmable vector interrupt controller

Table 5: Features of PIC32MX795F512L

Figure 21: PIC32MX795F512L

Figure 22: PICKIT3

PICkit version 3 is used as the programmer, providing low cost and stable in-circuit debugging functions with a minimum of additional hardware in need, shown in Figure 22.

MCP2021 is chosen as the LIN transceiver. It provides a bidirectional, half-duplex communication interface to automotive, and industrial LIN systems bus specifications 1.3, 2.0 and 2.1. The device incorporates a voltage regulator with (5V, 50mA) or (3.3V, 50mA) regulated power supply output. The regulator is short circuit protected and is protected by an internal thermal shutdown circuit.

The Tx and Rx ports of the MCU are connective, enabling the capability of the master task and the slave tasks running simultaneously on the master node from a hardware's perspective.

3.2.2 Software selection

The LIN cluster's construction is based on C language and the development is processed by MPLABX integrated development environment. MPLAB X is the latest edition of MPLAB IDE built by Microchip Technology and is based on the open-source NetBeans platform. It supports editing, debugging and programming of Microchip 8-bit, 16-bit and 32-bit PIC microcontrollers. In this thesis, the XC32 compiler is used for code building and the former version libraries from the C32 compiler can be referred as a legacy in the hierarchy to provide convenience when some exclusive functions are in need since XC32 is not very friendly to non-Harmony based project.

The additional software used in this project includes Ubuntu operating system version 16.04, Python version 3.6, and Microsoft Excel 2017 for data analysis.

3.3 Testing devices

A Cabin Air ECU is chosen to be used as the slave node in the LIN network. There is no interface for reprogramming and all the functions are internally packaged. An outlook of the ECU is shown in Figure 23. There are several sensor sections which can detect the inner air condition, reflecting as statistics types of data like voltage and current.

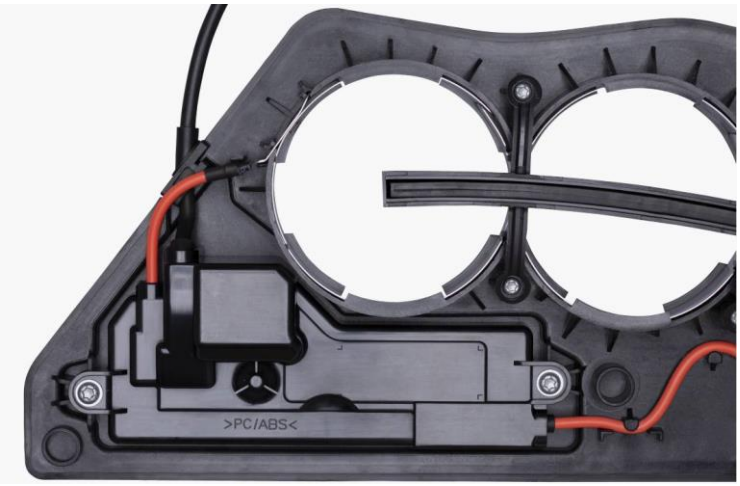

Figure 23: Cabin Air ECU

In this thesis, the ECU is regarded as a black box that supporting a variety of UDS functionalities. The LIN messages transmission can be detected by PicoScope 6, which is shown in Figure 24.

Figure 24: PicoScope

The PicoScope works as an oscilloscope to sniff the real-time signal exchange in the LIN network. The completeness of the LIN frame is an important factor that reflects the feasibility of the LIN model.

4 Implementation

This chapter describes the implementation process of the LIN communication. Section 4.1 introduces the system architecture and related hardware design. Section 4.2 shows the database we use to generate the LIN cluster in the project. Section 4.3 introduces the AUTOSAR software structure used in the overall design. Section 4.4 presents the LIN driver layer of the LIN model construction. Section 4.5 describes the LIN interface design and the LIN application layer design is presented in section 4.6. All the layers interact with each other with a reusable interface and provide a dependable connection between high-level services and hardware.

4.1 System design

In previous parts, the related hardware selections and software technologies have been introduced. This section gives a specific view of the connection of the components in the system architecture design. This method uses UART serial protocol as the hardware interface between MCU and LIN transceiver, the tasks performed on the port are limited in byte sending and receiving with featured dominant bit and recessive bit. The architecture of the system is shown in Figure 25.

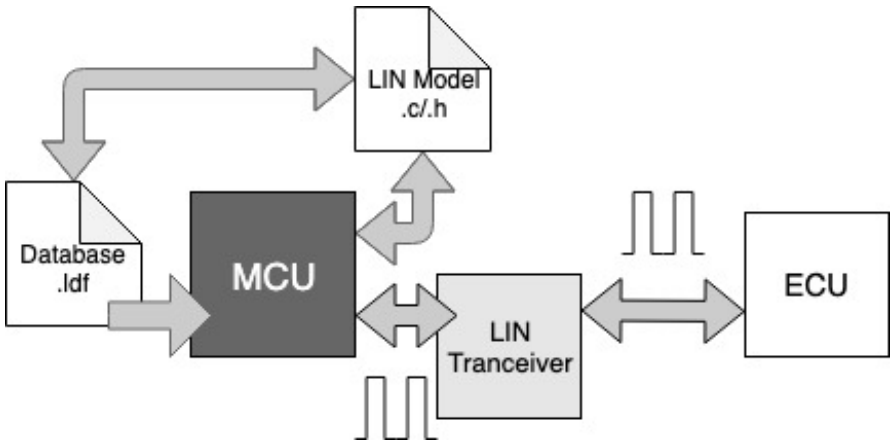

Figure 25: LIN system architecture

The LIN transceiver can provide a stable output by the circuit protection. The voltage of the UART output is not capable of driving the ECU, so a high-side circuit is designed to pull up the level to enable the LIN communication to ECU. The circuit design of the connection of LIN transceiver and ECU is shown in Figure 26.

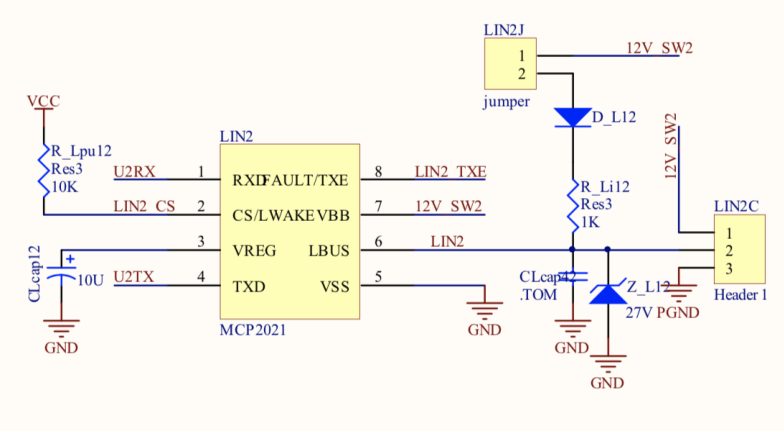

Figure 26: Circuit of LIN transceiver connection

The LIN system is powered by an external power source. To meet the physical of LIN communication and MCU platforms, a power adaption circuit is designed to adapt the input 12V voltage to 3.3V and 5V to drive the MCU and its peripherals. The power supply circuit is shown in Figure 27. PDT006 is used as a converter. The converter has the ability of over-temperature protection, remotely switching on and off and output overcurrent protection, and is designed with an input range of 3V-14.4V and output range of 0.45V to 5.5V, which can fully meet the requirement of the LIN system. The circuit provides power supply to the MCU's peripheral components. For the power supply of the MCU, device TPS75833KTTT is applied with an input range of 2.8V-5.5V and an output of 3.3V. The output voltage will then process several regulator circuits and relay circuits to gain a more stable power source.

The PIC32MX795 series MCU used in this project (shown in Figure 21) is general-purpose hardware platform, equipped with USB, CAN, and Ethernet interface. The operation condition of the MCU is under 2.3V-3.6V supply and

MIPS16e provides a code-efficient (C and Assembly) architecture. This project utilizes C language and UART to realize the LIN APIs and signal transmission.

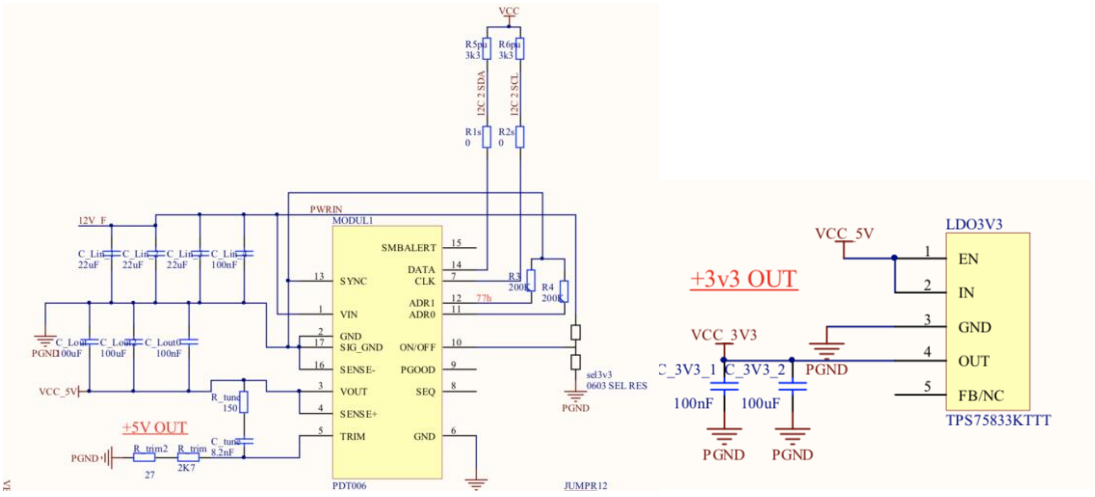

Figure 27: Circuit of power supply

The MCU has several power-saving features. When the CPU is running, power consumption can be controlled by reducing the CPU clock frequency, lowering the peripheral bus clock by individually disabling modules. Idle mode and sleep mode are also enabled to halt the CPU while the peripherals continue the operations. In this project, the idle mode and sleep mode of the LIN bus are used to reduce power consumption. The hardware system clock setting is configured as below.

```
#pragma config FPLLMUL = MUL_20, FPLLIDIV = DIV_2, FPLLODIV = DIV_1, FWDTEN = OFF
#pragma config POSCMOD = HS, FNOSC = PRIPLL, FPBDIV = DIV_1, WDTPS=PS512
#pragma config ICESEL   = ICS_PGx1
#pragma config UPLLEN   = ON
#pragma config UPLLIDIV = DIV_2
```

The system clock and baud rate configuration are based on the definition in the LIN database.

4.2 LIN database design

A LIN Description File database is used to describe the components with their properties, data flow, timing sequence, and hardware clock set. The frame transmission, speed of communication, values of signals, and the LIN protocol version can be easily changed in the database. The database is an independent

part over the top of the LIN basic workflow used in the thesis, shown in Figure 28.

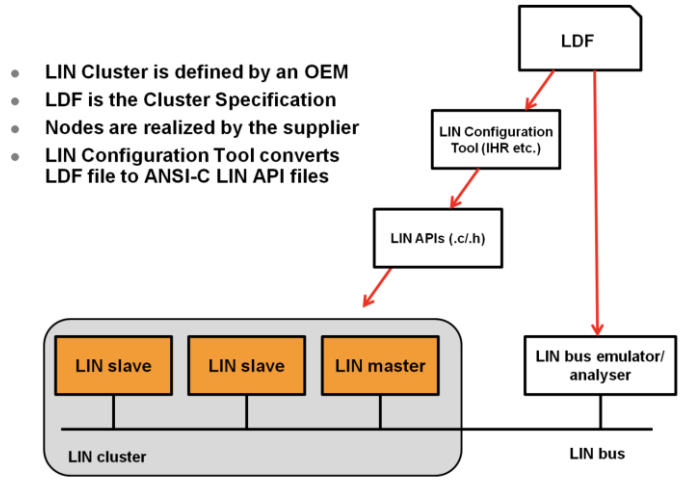

Figure 28: Standard LIN workflow

The database consists of a series of node capability files, which define the capability of each node in the cluster. According to the settings below, the node capability includes the supported LIN protocol, node address, product IDs from OEM, error definition, and configurable frames. The master node and slave node differ in the node capability in header transmission functions. The LDF database will be configured by a LIN configuration tool to generate a header file for future development.

```
Node_attributes {
    Demo_Slavenode {
        LIN_protocol = "2.2";
        configured_NAD = 1;
        product_id = 33,43605,60;
        response_error = Comm_error_1;
        P2_min = 0 ms;
        ST_min = 0 ms;
        configurable_frames {
            Demo_Command;
            Demo_Response;
                                }
     }
}
```

Based on the chosen use case, the main content defined in the database would be:

- To set the initial status of the LIN bus
- To define the composition of the LIN system
- To handle the start and stop of data transfer
- To handle a series of UDS signals transmission

The initial status of the LIN bus includes the initial value of signals and the physical layer setting. The data transfer process will follow the timing constraints and sequence defined in the schedule, and the UDS functions need another package in the diagnostic schedule. The database defines the properties of each node, the signals transmitting over the LIN bus, and the scheduling of each LIN frame, shown in Figure 29.

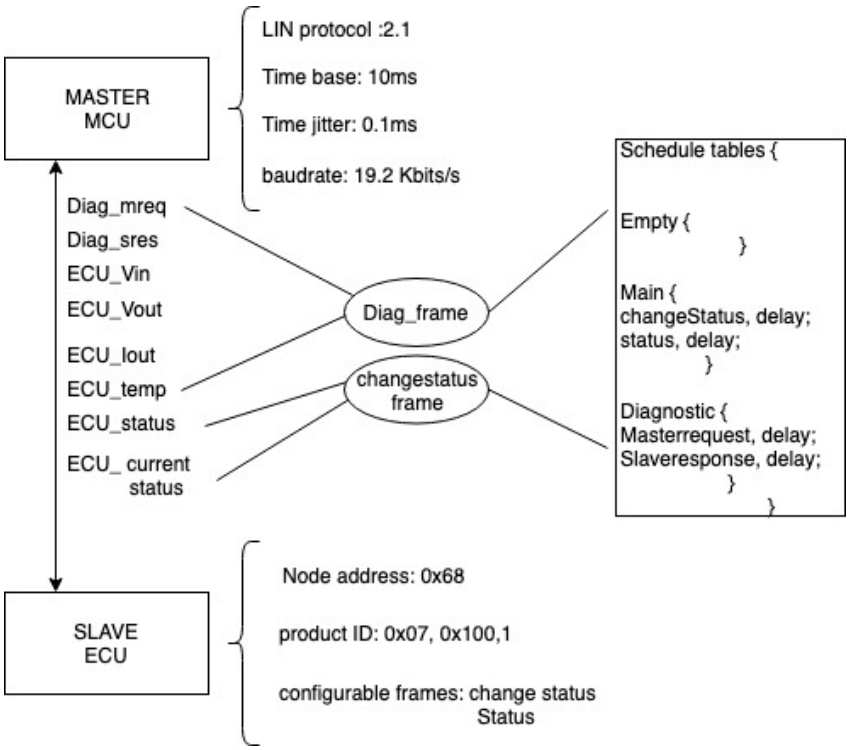

Figure 29: Node capability and signal properties in LDF

The LIN cluster designed in this thesis is composed of one master node and one slave node. The master node is capable of switching the status of the slave. Several statistic data are set in the ROM of the slave node by the manufacturer with a certain offset, which can be reached by corresponding diagnostic services in the UDS.

Data is transferred as a signal on the LIN bus in the types from Diag_mreq function to ECU's current status in Figure 29. The signals are packet into frames with specific PIDs defined in the frame table. And the transmitting sequence is set in schedule tables, the scheduling controller shall decide whether to switch to a new schedule or start from the first frame. The node capability describes some of the timing features and supporting LIN protocol on the LIN master, and defines the node address and product ID of the slave nodes, which are important in the UDS.

Based on the LIN workflow in Figure 22, the LDF will be translated into ANSI-C language header files (genLinConfig.h and genLinConfig.c in this thesis) to be loaded on the LIN application. The format of LDF is shown below.

- <LIN_description_file> ::= **LIN_description_file ;**
  <LIN_protocol_version_def> <LIN_language_version_def>
  <LIN_speed_def> (<Channel_name_def>) <Node_def>
  (<Node_composition_def>) <Signal_def> (<Diag_signal_def>)
  <Frame_def> (<Sporadic_frame_def>)
  (<Event_triggered_frame_def>) (<Diag_frame_def>)
  <Node_attributes_def> <Schedule_table_def>
  (<Signal_groups_def>) (<Signal_encoding_type_def>)
  (<Signal_representation_def>)

In this project, the physical features and LIN cluster composition are set as follows, with 19.2 kbps baud rate, 10 ms time base, and 0.1 ms time jitter.

```
LIN_protocol_version = "2.1";
LIN_language_version = "2.1";
LIN_speed = 19.2 kbps;
Nodes {
    Master : LIN_Master,10 ms,0.1 ms;
    Slaves : ECU;
}
```

As shown in Figure 29, the signals in the LIN cluster are predefined in the finite set, which enables the LIN protocol's predictability. For signals in unconditional frames, the direction of the transmission is defined to describe the publisher and subscriber. The length and initial value of the signals are presented in the database for memory allocation in the configuration process. For signals in diagnostic frames with PID 0x3C and 0x3D, the publisher and subscriber are predefined in the LIN protocol specification, dividing into master request signal and slave response signal. The values of signals can be changed in the LIN application, and the transmission process will take frames as carriers.

```
Signals {
    Vin :   8, 0, ECU1, MCU;
    Vout:   8, 0, ECU1, MCU;
    Aout:   8, 0, ECU1, MCU;
    assignECUStatus: 8, 0, MCU, ECU1;
    assignedECUStatus:8, 0, ECU1, MCU;
    Temp: 8, 0, Ecu1, MCU;
}
```

The frames definition in the database specifies the publisher, PID, and byte length of the data field. Multiple signals can be carried by the same frame, so the bit offset of each signal is also clarified. Diagnostic frames follow the standard format in the specification, so the definition is universal in different databases.

```
Frames {
    changeStatus : MCU, 0x33, 1 {
        assignECUStatus, 0;
    }
    Status: ECU1, 0x11, 1{
        assignedECUStatus, 0;
    }
}
Diagnostic_frames {
    MasterReq: 0x3C {
    ...
    }
    SlaveResp: 0X3D {
    ...
    }
    }
```

The frame transmission is described in the schedule table. The table gives a fixed sequential order of the frames in the list, and the timing interval between adjacent tasks is regarded as a timing constraint of the communication process. The schedule table can contain several sub-tables to separate the frames for different functions. The switching mechanism between tables should be implemented in the LIN protocol model, otherwise, the transmission index will return to the current table after its completion.

```
Schedule tables{
    DiagRequestSchedule {
        MasterReq delay 15.000 ms;
    }
    DiagResponseSchedule {
        SlaveResp delay 15.000 ms;
    }
    StatusCheckSchedule {
        changeStatus delay 10.000ms;
        Status delay 10.000ms;
    }
}
```

The availability of pre-made off-the-shelf slave nodes is expected to grow in the next years. If the ECUs are accompanied by a node capability description, it will be possible to generate both the LDF database and initialization code for the master node. In this thesis, the LDF can allow a dynamic LIN cluster configuration based on the functional modification on the top application

layer without involving the LIN model's inner architectures. The LDF database is shown in Appendix 1.

4.3 LIN model implementation

To apply the LIN database in the LIN network, the LIN model shall be designed as a layer structure to provide reusable interfaces for different applications defined by LDF and various ECUs. According to the related work, the AUTOSAR-based structure can provide a more concrete and flexible structure with AUTOSAR APIs. In this project, the LIN model design derives from the AUTOSAR basic software layers with LIN APIs.

AUTOSAR is a software platform developed as a solution for the software demands in automotive embedded systems [18]. AUTOSAR is an open and standardized software architecture for vehicle software. It is developed jointly by automotive manufacturers (OEM's), suppliers and tool vendors working in coordination. AUTOSAR is dedicated to ECUs, which can also act as a LIN master.

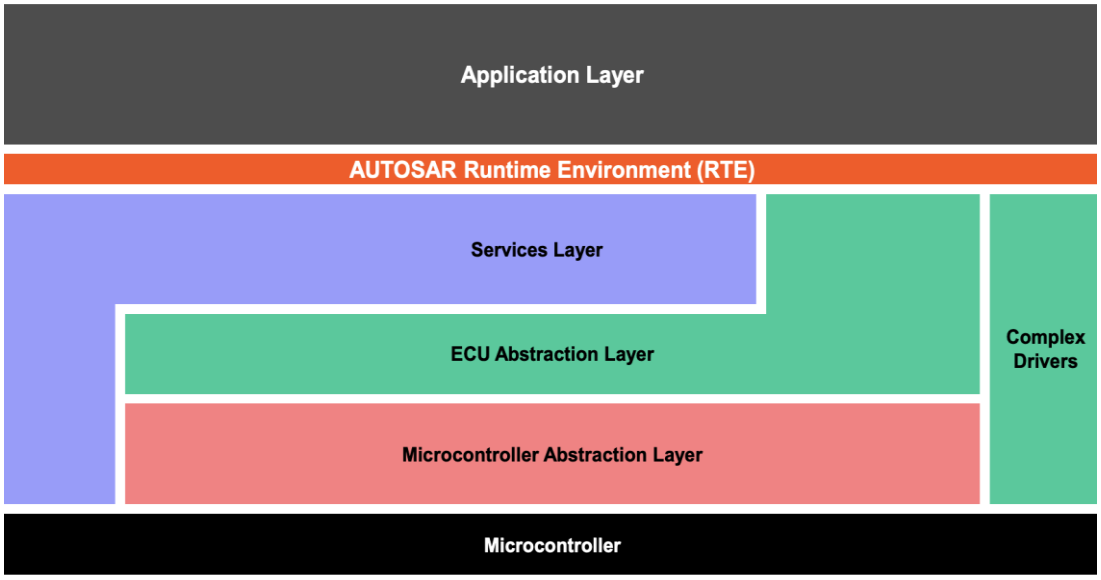

Figure 30: AUTOSAR software layer architecture [18]

The AUTOSAR architecture has three main objectives. The first is to build a hardware-independent layered software architecture; the second is to provide

a methodology for the future automotive electronics architecture, and the other is to develop various vehicle application interface specifications as an application integration standard for the reuse of software component on different automotive platforms. The layer structure is shown in Figure 30, and a specific description of the basic software layer is shown in Figure 31.

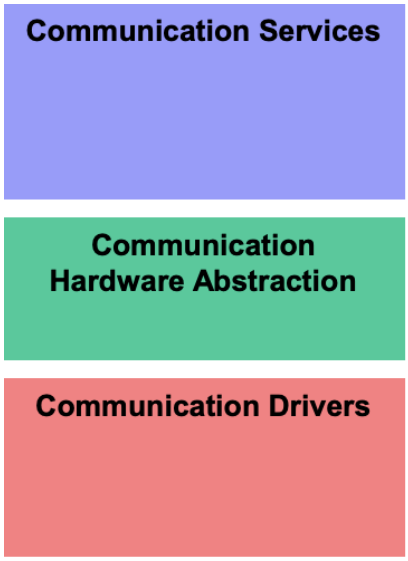

Figure 31: Basic software layer

Basic software layer consists of the services layer, ECU abstraction layer, and microcontroller abstraction layer. It can provide service options for software components in the upper layer instead of performing actual operations. The MCU abstraction layer can separate the software from coupling with a specific type of hardware, and provide the driver of the peripheral in the MCU with the memory mapping information. The service layer includes a series of memory services for regulating the memory access, communication services for signal transmission, system services for operating system services, I/O services, and complex drivers for providing drivers for complex sensors and actuators.

To apply the AUTOSAR in LIN, the layered architecture has been divided into LIN communication services, communication hardware abstraction, and communication drivers and hardware, depicted in Figure 32. The scheduler manager and its interfaces are used to decide the point of time to send a LIN

frame, and the LIN interface controls the wakeup/sleep API and allows the slaves to keep the bus awake. The PDU router accesses the LIN interface on the PDU level instead of the signal level. When sending a LIN frame, the LIN interface requests the data for the frame from the PDU router at the point in time when it requires the data (i.e. after sending the LIN frame header).

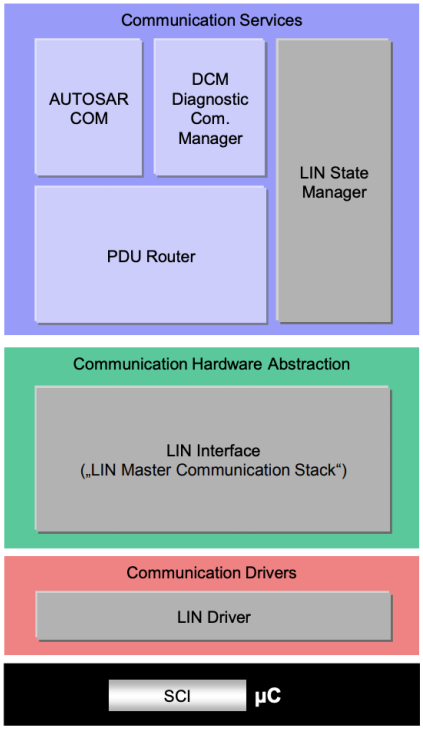

Figure 32: Basic software layer in LIN [18]

LIN communication services contain communication (COM), diagnostic communication management (DCM) [18], LIN state manager (LSM) [26], and protocol data unit router (PDUR) [19]. In this thesis, DCM is used to handle the UDS frames for sending the frame to the specific software section and regulate the process of the diagnostic, enabling the error code reading and restarting functions. PDUR is a connection between upper schedule to lower model, and unpack the frame received for the transport layer. LSM manages the basic status of the bus such as sleep or awakens, as well as the initial schedule's setting.

The DCM module within the AUTOSAR framework provides common program interfaces for diagnostic, configuration, and identification services. It can ensure the diagnostic data flow and regulate the inner states. The DCM also has the capability to be network-independent. The network-specific tasks can be handled outside of the module, with the interface provided by the PDUR module.

The PDUR module provides functions to transmit and receive diagnostic data. It provides DCM module with incoming diagnostic requests and several operations of the DCM are based on the precondition that the PDUR interface supports all service primitives defined for the service access point between the application layer and underlying layers.

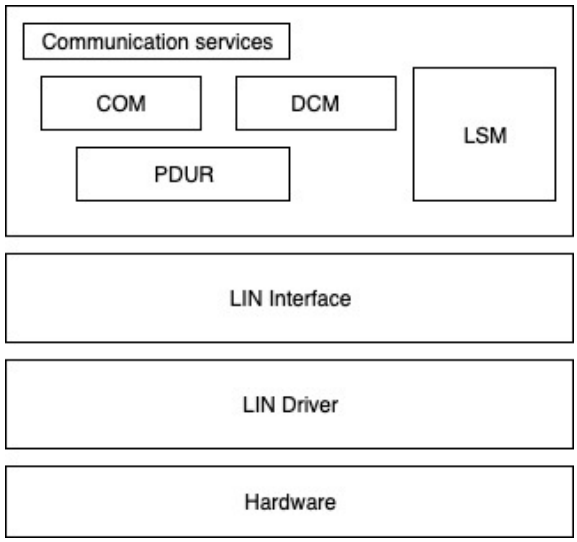

Figure 33: LIN AUTOSAR model

The model provides a uniform interface to the LIN network, hiding protocol and message properties from the application. The overall header file structure is shown in Figure 33. The LIN driver, LIN interface, PDUR, and other models are packaged into APIs in the header files, which can separate the top-level application and hardware, enabling the feasibility to replace the system with other types of hardware (ECU) or UDS services [14].

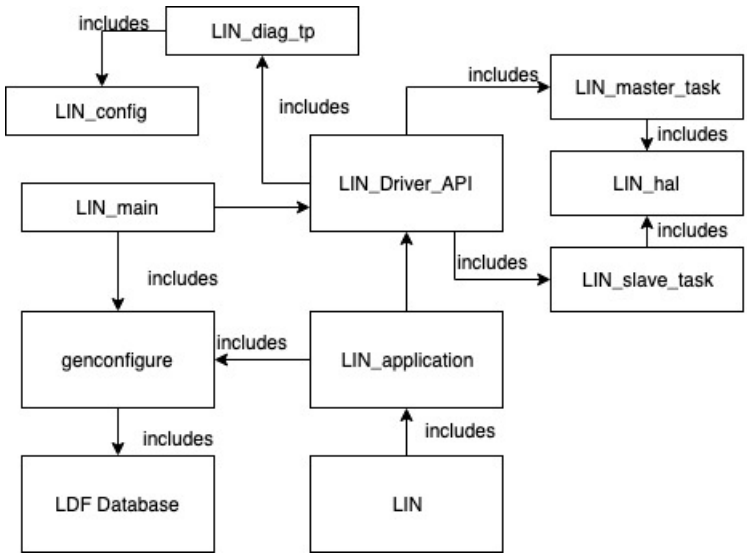

Figure 34: Header file structure of the LIN network

Shown in Figure 34, LIN_driver_api function designs the APIs in the LIN state manager model, including the table scheduling and sleep command, and an error will be generated if the return state is inconsistent. LIN_slave_task function and LIN_driver_api function define the functions of the LIN Protocol Data Unit Router, decompose the identification of the frame, and transfer the units to LIN diagnostic communication management which is defined in Lin_config function and LIN_diag_tp function. All the behaviors of the APIs in the function models are defined in the genconfigure.c file from the LDF database.

4.4 LIN driver design

The driver serves for the ECU layer and LIN interface layer. LIN driver is part of the microcontroller abstraction layer, providing a hardware-based interface for the upper layer. The operations of the LIN driver can happen on the entire LIN driver layer or a single LIN channel. The initialization and awakening of the LIN bus are done by the driver, which is effective on both channels. The other functions like channel initialization, frame header transmission, data (response) transmission, and go-to sleep command are included in the single-channel operations.

According to the definition of the LIN driver in AUTOSAR, the driver will interact with the ECU State Manager starting from the LIN_init API model. In this thesis, the initialization model resets all the state bits for the transmission process and data buffer. The operations are hardware-specific, and varying with various hardware platforms. The LIN_init function consists of local timer resetting, status byte resetting, and uart_init function initialization service. It can also provide the entry of an empty frame schedule table to set the LIN system to an idle state. The core code is shown below.

```
lin_hal_wait_for_break();
lin_timeout_ctrl.flag.sys_init = 1u;
lin_hal_init_uart();
lin_timeout_ctrl.flag.sys_init = 0u;
```

The initialization APIs also provide the channel setting for the upper layer with the configuration of baud rate, interrupt signals and other relevant control variables.

4.4.1 LIN master task

Acting as the interface of the LIN interface layer, the LIN driver can be divided into a master task and a slave task, used for header transmitting and response transmitting respectively. The state machine of the master task is shown in Figure 26. With the names of API defined in the LIN specification 2.1, lin_master_tx_header function is used to send the header onto the LIN bus. As defined in the LIN specification [1], the function includes several sub-functions to transmit the break field in a lower baud rate to ensure the synchronization, the sync byte, and the protected identity. The function will not transmit the header at once, but load the header into a static buffer. When the next interrupt is due, the header will be sent to start a new frame transmission.

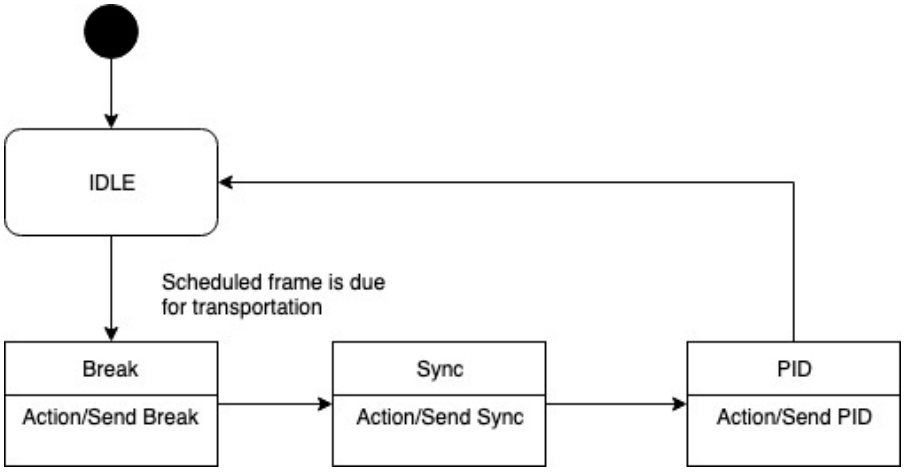

Figure 35: State diagram for header transmission

The conversion between two adjacent states is occurred based on a status check. The state machine calls a status-request function to acquire the current status of the LIN transmission. The initial status is always set idle as default, then the state machine starts the status checking for the next signal departure. If the local timer detects a timeout state, the process will be initialized, i.e., the state is idle, and then the status can transfer to break_received, sync_received and PID_received states. In this thesis, to guarantee the correctness, the master task is built upon the state checking of the receive port of the master, since the master will obtain the signal sent by itself (according to the hardware structure of the LIN transceiver MCP2021). The flow chart is shown in Figure 37.

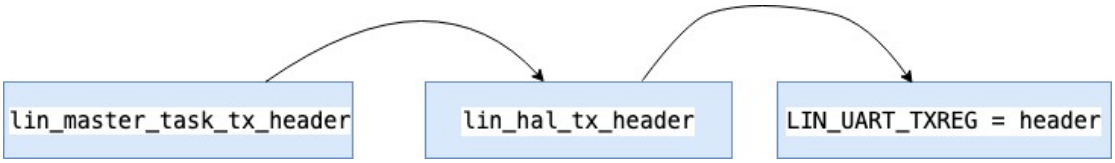

Figure 36: Mapping process of LIN transmission

The hardware mapping of the header transmission results on the UART byte transmission, from the software layer to the hardware layer via a hardware abstraction layer, shown in Figure 36.

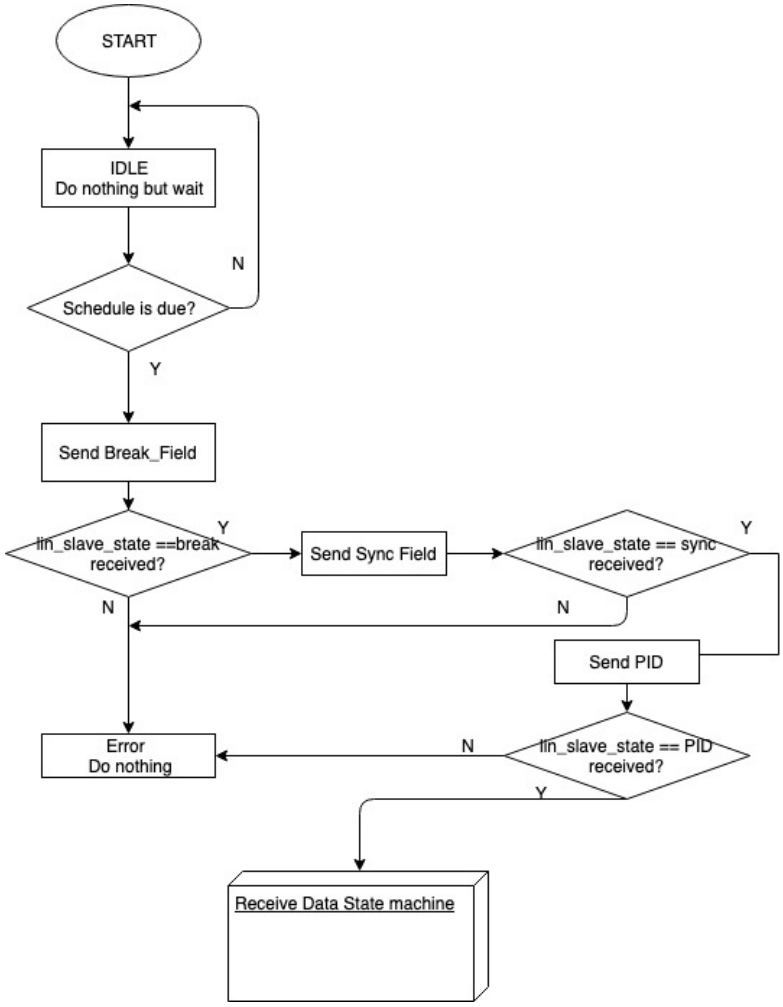

Figure 37: Flowchart of master task

In the figure, Lin_slave_state describes the state of the response (slave task), and each section of the header information is regarded as the response of the former signals. After the PID received, the state machine shall transfer to the receive_data_state_machine, which acts as a router for transferring different kinds of frames to the upper layers. If there is no scheduled frame, the LIN system will stay in an idle state. The error is generated when the expected state fails, the current transmission will be aborted and the system should wait for another schedule trigger. The LIN master task is the essential initial part for every frame in the schedule table, the other data section will not be sent or received if the current state is not PID_received. The type of a certain frame is defined in the LDF database and reflected as a flag bit in the schedule

structure, so the correctness will not be threatened by the combination of the slave state and master task.

4.4.2 LIN slave task

LIN slave task includes response reception and response transmission. The two functions are switched due to the received PID. Each PID defines a certain direction of the frame transmission in the LDF database, and only the predefined static data buffer is visible to the following communication operations. The slave task includes two state machines:

- Break/sync field sequence detector, which is also used in the master task in this thesis, without transmitting actions. The strict sequence is ensured by the task. These state machines are exclusive in the master node.

- Frame processor to process frames with various functions, to transmit or load data from or to the local buffer. In the LIN driver level, the types of LIN frames are limited in sending frames and receiving frames, which can be handled by the hardware-related UART operations. The advanced classifications of the unconditional frame and diagnostic frame are used in higher layers.

Similar to the master task, the LIN slave task is based on the reception operations on UART. The driver checks the states of the receive registers of a series of UART ports and the interrupt service routines are used to read the data from registers to the buffer. The hardware mapping in slave tasks is to verify the property and identity of received data from the register monitoring.

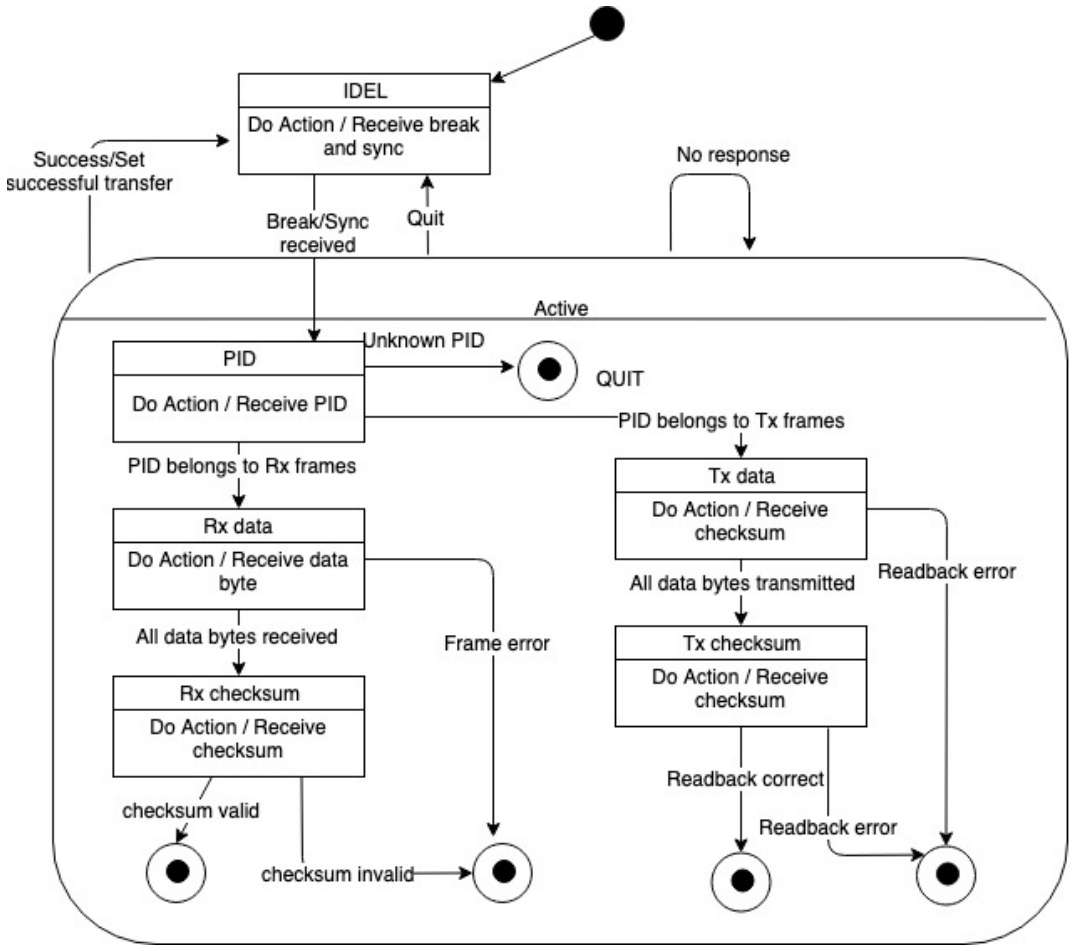

Figure 38: Frame processor state diagram

Figure 38 describes the state conversion in the slave task. The frames are roughly divided into two types according to the PID: Tx frames and Rx frames for transmitting and receiving data respectively. The transmitting and receiving only affect the buffer, regardless of the specific function of the current frame. Both processes shall end with a correctness check of the checksum, and the state machine will jump to idle state if an error is detected at any state. The error handling mechanism varies from different error types. For service error, the LIN system will reset the communication state machines of the current process to repeat the task; when time out error occurs as the task cannot recover, the system will skip the fault and continue other scheduled tasks. When checksum error occurs after the data field transmission, the driver will abort the task, and the checksum will be

recalculated in the next loop. The algorithm of checksum calculation uses enhanced protocol in this project.

```
static void l_hal_checksum_calculation(void)
{
   l_u16 tmp_sum;

   tmp_sum = (l_u16)((l_u16)lin_frame_data_g.checksum + (l_u16)lin_hal_rx_data_guc);
   if (0x00FFu < tmp_sum)
   {
      lin_frame_data_g.checksum++;
   }
   else
   {
      /* Do nothing */
   }
   lin_frame_data_g.checksum += lin_hal_rx_data_guc;
}
```

The L_CTRL_TYPE_OFFSET flag indicating the publish or subscribe property and predefined in the user-defined database, has direct connectivity with the PID estimation in the state machine. The LIN driver will call different interrupt service routines when a signal received. The flag in the register U2RXIF will be modified and reset when the ISR is called.

4.4.3 Sleep and wake up command

Network state management in a LIN cluster refers to cluster wake up and go to sleep only. The state diagram in Figure 39 shows the behavior model for the communication of a slave node.

The initializing state is instantaneously entered after the first connection to a power source, reset or wake up. The operational state indicates the node is ready for frames transmitting and receiving and the sleep node only allows the wake-up signal on the bus.

The sleep command is the frame with data field: 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF. Wake up signal is a dominant pulse longer than 150 us.

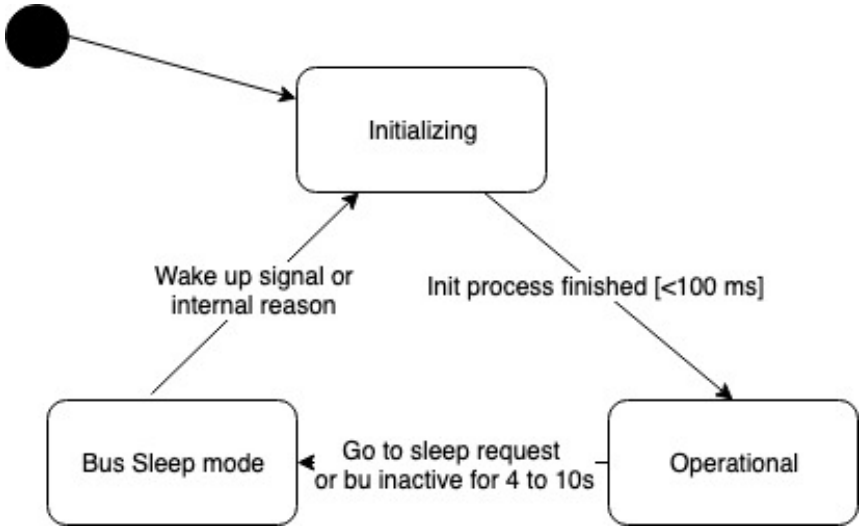

Figure 39: Slave node communication state diagram

When the current scheduled frames tables finish, the system will wait for the go-to sleep command or the timeout of a local timer to disable the transmission function. The reception function will be always enabled to wait for the wake-up signal.

According to the above description, the LIN driver designs the interface to the interface layer and hardware layer respectively, the dependency of the external port, and possible process states management.

4.5 LIN interface and communication services

In this thesis, the LIN interface manages the schedule tables and process frames, which are divided into unconditional frames and diagnostic frames. The unconditional frames manage the data writing and reading on the ECU, whose data section is composed of up to eight data. The diagnostic frames are the frames with the PID of 0x3C and 0x3D, including diagnostic functions, node configuration functions, and node identification functions whose data section will be processed by the router and following diagnostic message manager.

4.5.1 Schedule management

LIN interface uses schedule table mechanism, with every channel corresponding to every table including multiple frames.

LIN interface regulates the sequence of the schedule defined in the LDF database. The interface keeps running the schedules to enable lower layer performing frame header transmission. No interrupt shall occur during the process of any schedule, the schedule management will transfer to the first frame of another schedule if a new task is set, otherwise, it will restart the transmission of the current schedule.

4.5.2 Transport layer frame process

According to the LIN specification [1], the frames can be classified into unconditional frames, event trigger frames, sporadic frames, diagnostic frames, and user defined frames. Diagnostic frames include master request frames (MRF) and slave response frames (SRF). The connection between transport layer and application layer is shown in Figure 41. The frame process classifies frames into a TP frame and non-TP frame (unconditional frames and diagnostic frames in this thesis). Only the TP frame has the necessity to request a room for a buffer from PDUR.

Before the transmission of single or multiple frames, the schedule management defined in the LIN interface inserts the target table to the processing list.

For unconditional frames（non-TP frames）, the inner communication is not transport layer included. The data field does not satisfy the format of a PDU, so the frame process does not have to unpack the frame with section information to further processing. The communication goes directly between the signal processing application and the signal carrying frame.

The main loop calls the schedule table in the LDF database to find a proper frame to start the process and accomplish the header transmission, and the received PID determines the following operations. If the PID maps to a

transmitting frame, the LIN engine will request the response data for transmission and status check from PDUR; if the PID maps to a receiving frame, the engine will load the received data to the PDUR.

In this thesis, the LIN master can both transmit and receive the unconditional frame, which are change-Status and Status frame in the LDF database. The data receive state machine starts for the rx_data status in the master task state machine, shown in Figure 40.

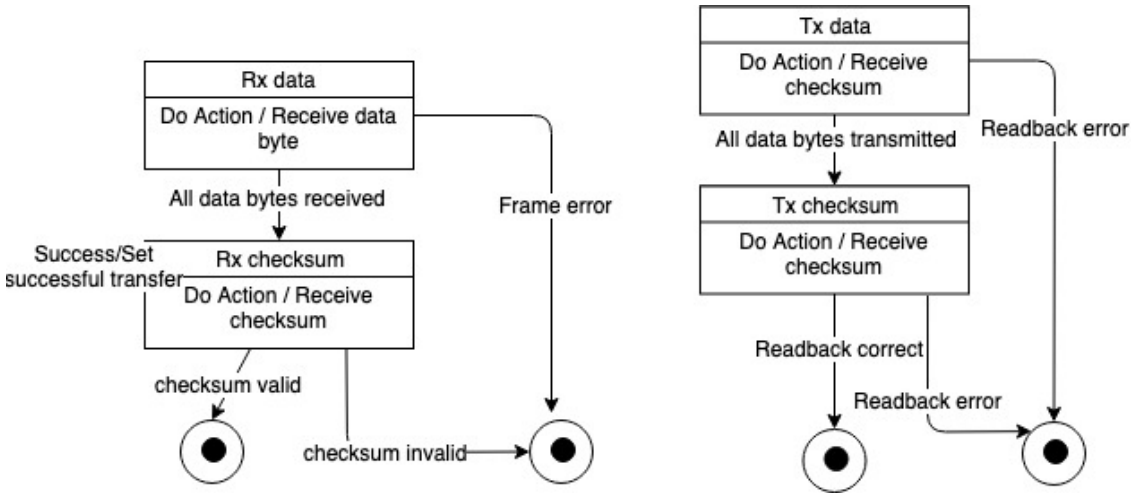

Figure 40: State machine of signal carrying frame process

The signal received shall be stored in the receive buffer of a certain hardware in a uniform format. And the data for transmission shall be shifted to the port from the buffer, ending with a checksum.

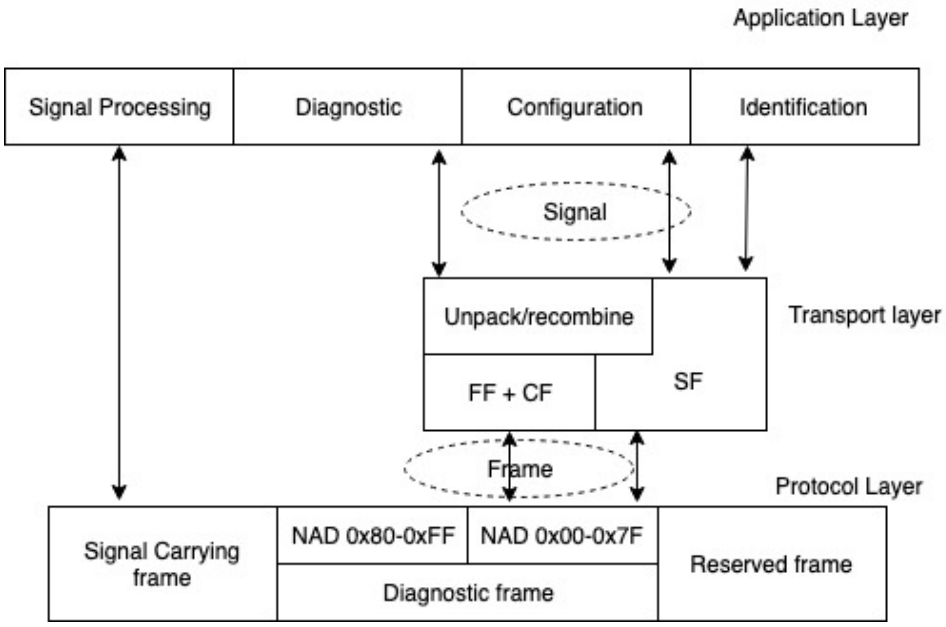

Figure 41: Transport layer in LIN model

Before sending the header, the engine should load the message into the buffer in PDUR, the TP messages are a serial of data signals with same PID, so the length of the messages is not constant, and the data field can exceed eight bytes. In this project, identification services are used, according to the LIN specification [1], only a single frame (SF) is used and multiple frames first frame (FF) and consecutive frame (CF) are not adopted.

For diagnostic frames (TP frames, identification services with diagnostic header), the transport layer shall compose and decompose the signal in each frame to identify the target node address, information length, and service ID. For the transmission process, the frame should provide specific information on the slave node.

NAD	PCI	SID	Id	Supplier id-low	Supplies Id-high	Function Id-low	Function Id-high
0x68	0x06	0xB2	0X38	0X07	0X00	0X00	0X01

Table 6: Frame of the UDS in the thesis

Table 6 shows the frame structure of one of the UDS frames used in the thesis. When the application layer schedules a UDS function request, the transport layer shall packet the signal into the diagnostic frame sections. The NAD, supplier ID and function ID exclusively defines the node properties which is set by the manufacturer. The PCI stands for the length of the data field (SF in this thesis). The UDS supported by the selected ECU is shown in Table 7.

Hex id	Name
0x38	Supply voltage
0x39	Output voltage
0x3A	Output current
0x3B	PCB temperature

Table 7: UDS for identification services in the thesis

The state machine of the transport process is shown in Figure 42. When the receiving state machine detects the state of diagnostic PID, it shall enter the transport layer state machine. The PID differs in the unconditional frame and diagnostic frame. The PDUR distributes the frames to other LIN communication services based on the PID. When unconditional frames are loaded into the PDU router from the interface, the system will trigger signal interaction service to implement the data restore and fetch operations. The DCM will provide the UDS services if diagnostic frames are recognized in the router. The core process of the communication is loading the buffer with UDS signals. The UDS is predefined in the application layer, and the communication services will fill the diagnostic frame when it is due.

```
LIN_DIAG_SEND_FRAME.byte[0] = NAD;
LIN_DIAG_SEND_FRAME.byte[1] = 0x06u;
LIN_DIAG_SEND_FRAME.byte[2] = 0xB2u;
LIN_DIAG_SEND_FRAME.byte[3] = id;
LIN_DIAG_SEND_FRAME.byte[4] = (l_u8)(supplier_id & 0xFFu);
LIN_DIAG_SEND_FRAME.byte[5] = (l_u8)((supplier_id >> 8) & 0xFFu);
LIN_DIAG_SEND_FRAME.byte[6] = (l_u8)(function_id & 0xFFu);
LIN_DIAG_SEND_FRAME.byte[7] = (l_u8)((function_id >> 8) & 0xFFu);
g_lin_config_status = LD_SERVICE_BUSY;
g_lin_config_rbid_data = (l_u8 (*)[])data;
```

The configuration status will be set busy to trigger the responding communication services.

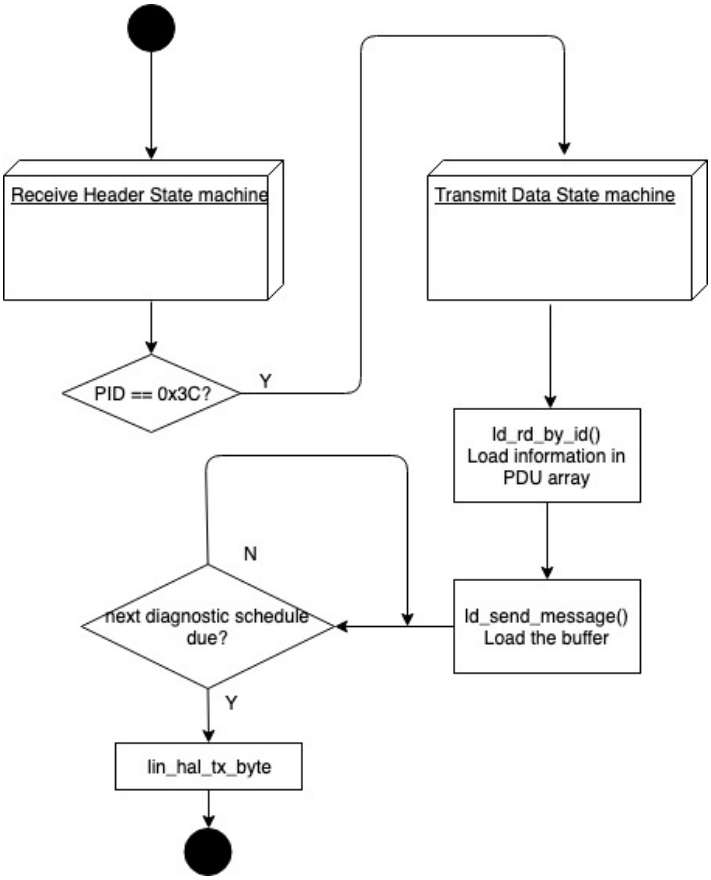

Figure 42: Flowchart of diagnostic message transmission

All the data transmission shall be done in lin_hal_tx_byte function, which operates the register directly. The UDS APIs in the thesis perform the data loaded from the application layer or LDF database to the local transmission buffer.

When receiving a diagnostic response, the transport layer shall unpack the frame to decompose each section for node and service information. The flowchart of the diagnostic message reception in this thesis is shown in Figure 43.

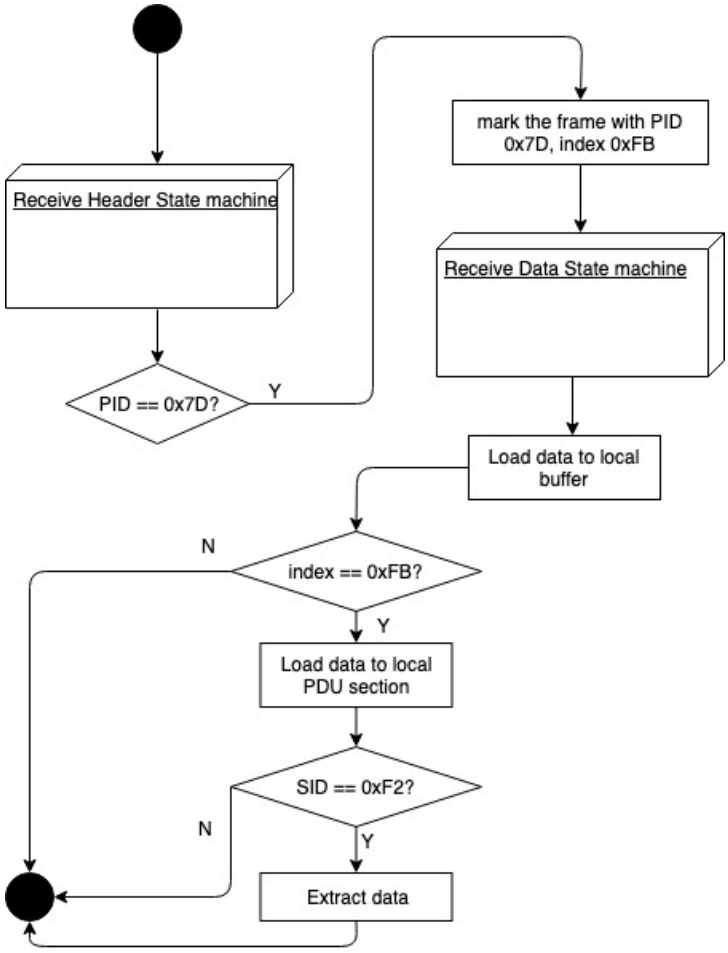

Figure 43: Flowchart of diagnostic message reception

When a header of diagnostic response (0x7D) is detected by the state machine, the LIN interface shall mark the frame with a certain index for future process. Then the data field will be loaded into the local buffer on each UART interrupt service routine. When the transport layer detects the diagnostic index, the data will be recomposed into one single frame packet data unit whose sections are segmented into NAD, PCI, RSID, and data. Then the interface will send the response data to the corresponding service ID according to equation 3. The target data indicating the identification of the ECU will be extracted by the application layer.

The LIN interface layer provides interfaces to the upper layer services such as LIN state management and LIN PDUR. Three interfaces are provided to the LinSM by the interface layer. The sch_set function is provided for requesting a

new schedule in the processing list, the l_ifc_goto_sleep function allows the state management to stop the entire LIN network with a master request frame, and the wake-up port to enable the cluster with l_ifc_wake_up function.

The interface layer provides the support for the transmissions of sporadic frames. If sporadic frames are enabled, the engine will record the information of the frame and finish the communication when the schedule is due.

The interface layer is also connective with the Basic Software model service. Sch_tick function enables the periodical calling of the schedule models, and the state switch among the frame transmission, schedule switching, and the LIN bus state management. The type of main data structure like the frames, PDU, and the schedule entries are described in the LDF database in appendix A.

4.6 Application layer design

As the top level of the LIN model, the application layer defines the behavior of the LIN communication independently, and different applications indicate various signal processing together with diagnostic services, which both are supported by the LIN interface and LIN driver. With LIN APIs, the application layer provides the network management services, signal carrying frames schedule, and diagnostic frames schedule from a high-level perspective.

According to the schedule table in the LDF database, the state machine of the transmission of frames is shown in Figure 44.

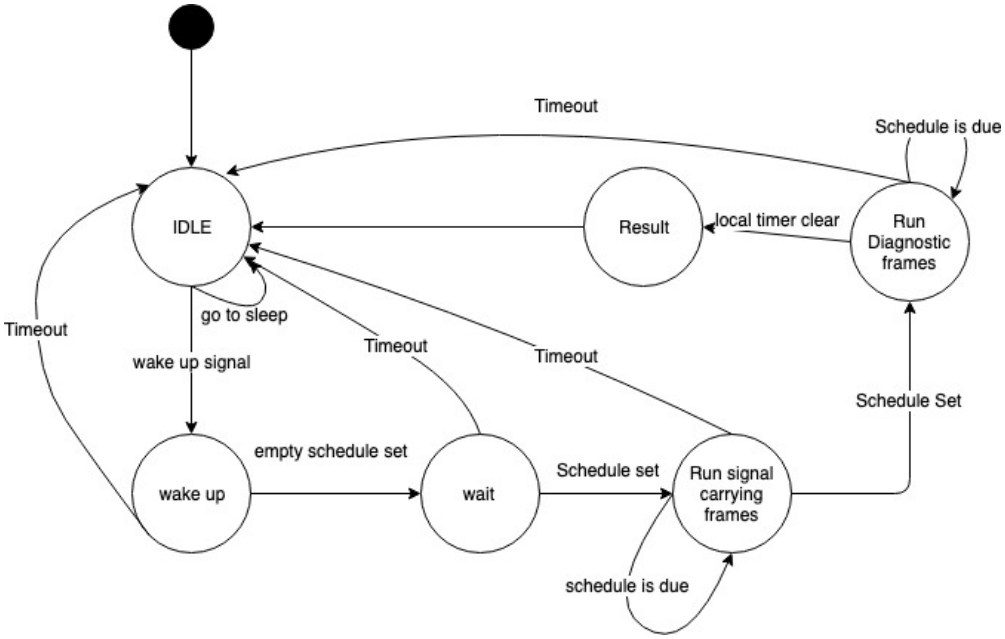

Figure 44: State diagram of the application layer

The application layer set the LIN system idle in the first stage, the interface and driver will provide an entry to an empty schedule. State management will trigger the LIN bus and start with the entries of frame schedules. The signal carrying frames will continue running if no switching occurs. In the section of running diagnostic frames, the configuration information can be replaced manually outside the loop, so various UDS functions or diagnostic messages can be finished in the identical schedule. For example, the read-by-id UDS services' parameters can be replaced with the method shown below:

- ld_read_by_id (0x68u, 0x0007u, 0x0100u , 0x38u , response_bytes)
- ld_read_by_id (0x68u, 0x0007u, 0x0100u , 0x39u , response_bytes1)

The two services are applied with the same hardware properties (node address 0x68, supplier ID 0x0007, and product ID 0x0100), and different service IDs to diagnostic the output voltage and current of the ECU.

The core LIN implementation in the software layer is well packaged into a simple loop. Unless a block generated by external signals, the LIN system will not stop running. The main loop in the core applications is shown below:

```
for(;;)
{
  /* 1ms cyclic task */
  if(0u !=  Timer1msTick)
  {
    Timer1msTick = 0u;
    #if ((defined LIN_SLAVE) || (defined J2602_SLAVE))
    LinSlave_Task();
    #endif /* #if ((defined LIN_SLAVE) || (defined J2602_SLAVE)) */
    #if ((defined LIN_MASTER) || (defined J2602_MASTER))
    LinMaster_Task();
    //cal();
    #endif /* #if ((defined LIN_MASTER) || (defined J2602_MASTER)) */
  }
```

In this thesis, four sub-states of configuration types are contained in the run diagnostic frames state by setting the data manually together with the diagnostic headers transmitting in the background. In every sub-state, the LIN interface shall return the current status of the service to maintain the integrity of every process, and the state conversion is performed in the configuration service state machine. These statuses inside the communication are included in every task, and the execution of tasks conversion is based on a state machine as well.

5 Result

This section describes the results of the prototype implementation and discusses them. Section 5.1 describes the result of sequential LIN communication. Section 5.2 shows the feasibility and performance of the designed LIN model with the proposed use case. Section 5.3 presents the generated configuration file from the LDF database use case in the application layer and the transmitting message construction results with resource consumption. Section 5.4 shows the UDS performance on the designed LIN model's use case in the running LIN cluster.

5.1 Result of sequential method

The sequential method consists of a series of operations on the serial ports of the master node. Without a layered structure, the signals in a frame are transmitted sequentially onto the LIN bus. The correctness can be guaranteed manually at the expense of low flexibility because of merging application code with hardware abstraction description. Thus, the transmission process can be regarded as an evaluation criterion in this thesis to judge if the LIN frame is completely and correctly filled. The control flow of the sequential method is depicted in Figure 45.

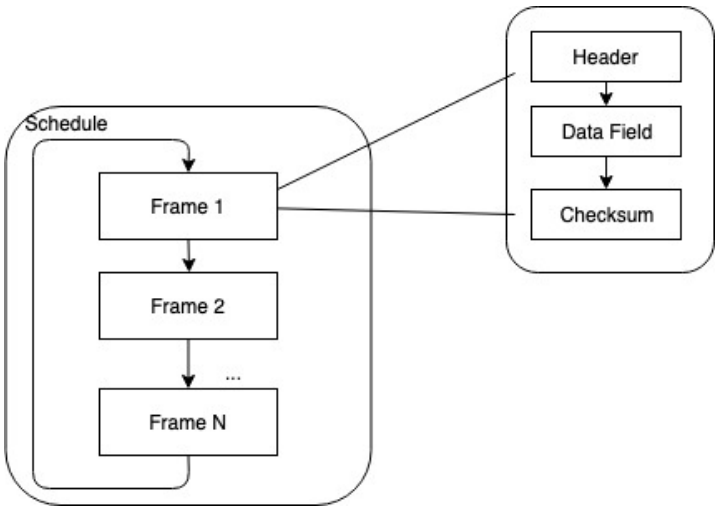

Figure 45: Work flow of sequential LIN communication

The sequential method does not apply a database to describe the network from a high-level perspective, and no timing constraint will be predefined. The resource consumption of compilation is shown in Table 8.

RAM	160 Bytes / 131072 Bytes
ROM	17196 Bytes / 536560 Bytes

Table 8: Memory consumption of sequential communication

The communication can be detected by the PicoScope, shown in Figure 46.

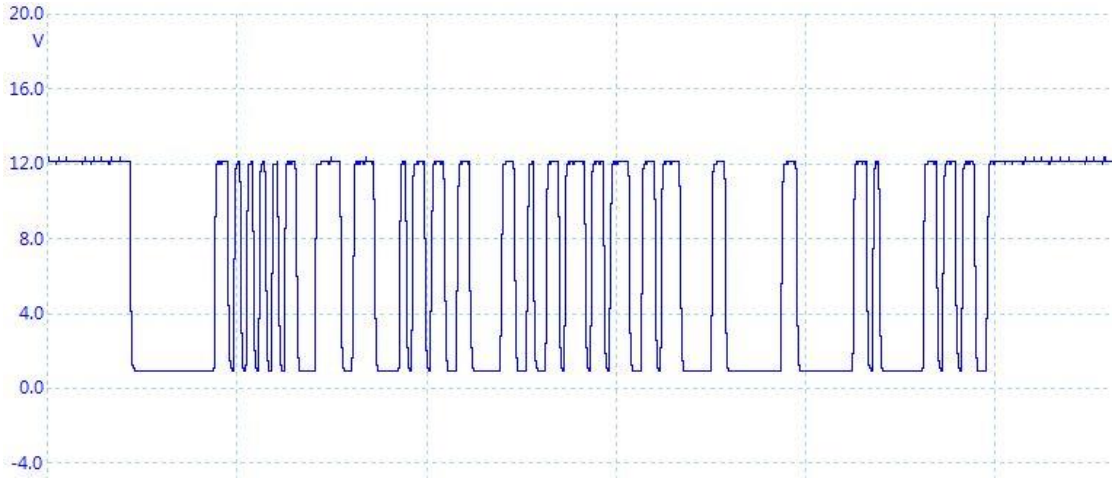

Figure 46: Sequential LIN communication

The results indicate that the sequential method is capable of running on the selected MCU. The transmissions of the header, data field, and checksum are tested to be correct by setting in the LIN transmission loop.

5.2 Results of LIN model tests

Test cases from the functions defined in the UDS standard are selected to test the LIN model's capability. A global-variable testing buffer to receive the return signal to indicate the feasibility of the services. The test cases differ in the SID section in the Diag_send_Frame function's structure. Every SID will be responded by an RSID case in the state machine in the protocol data unit router.

The LIN model is constructed according to the basic software layers of AUTOSAR to provide communication services. All the services are realized by

the cooperation of the sending state machine and the receiving state machine. The sequential method can only be used to implement the master node. No reception check will be performed to respond to a certain service with an RSID. As a result, the data dump service is not supported by the sequential method.

Function	LIN Model	Sequential Method
Assign NAD	✔	✔
Assign frame ID	✔	✔
Read by ID	✔	✔
Conditional Change NAD	✔	✔
Data Dump	✕	✕
Reserved	✔	✔
Save Configuration	✔	✔
Assign frame ID range	✔	✔

Table 9: UDS capability of the LIN model

Since the LIN model is designed based on the latest LIN specification, the data dump service is defined to be only supported by supplier diagnostic and not in a running cluster, e.g. at the car OEM. On the other hand, the service with SID 0xB4 is reserved for the initial configuration of a slave node by the slave node, and the format of this message is supplier specific, which is not available in

the thesis. The reserved services will not be supported by the model due to its unpredictability, thus no definition of the RSID can be made in the receiving state machine.

To estimate the resource consumption of the designed LIN model, several different test cases are used for the measurement. In all test cases, the baud rate synchronization is disabled since the work is based on a master node in the LIN cluster. Including different features of checksum and diagnostic services, the LIN model API supports both LIN 2.X and LIN 1.3, the comparison between the two protocol versions shows the differences, and a bar chart will be provided to give a clear view.

TC Nr.	Configuration	Nr of frames	Diagnostic API	ROM(Bytes)	RAM(Bytes)
1	LIN 2.1 slave	2	None	10248	184
2	LIN 1.3 slave	2	None	8112	188
3	LIN 2.1 slave	2	Used	11352	224

Table 10: Memory consumption of the LIN model

As for the target UDS functions in this thesis, the compiled engineering file occupies 500 bytes of RAM and 21528 bytes of FLASH, which is capable of running on the selected MCU. The comparison of different methods in memory consumption is shown in Figure 47 and 48.

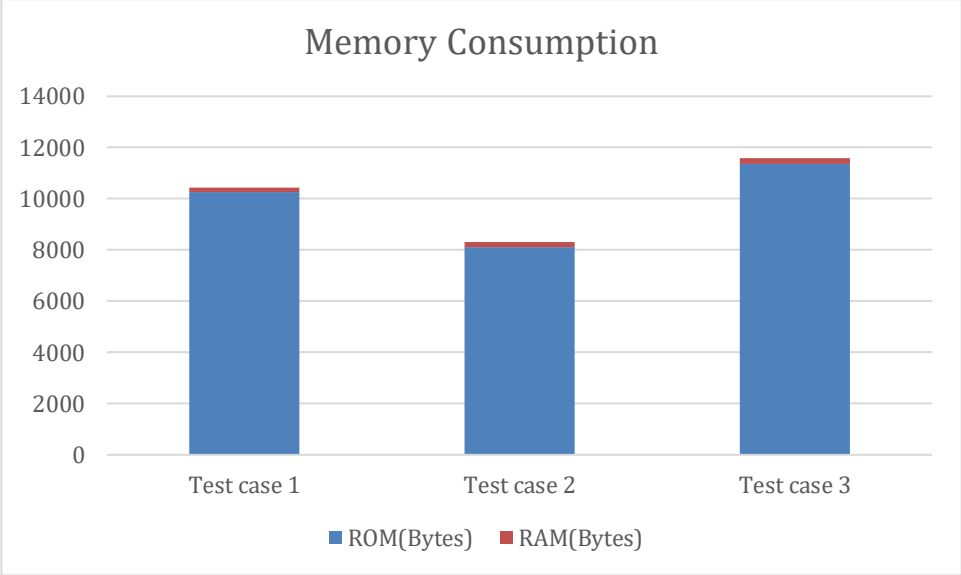

Figure 47: Memory consumption of the LIN model

Shown in the bar chart, the former version of the LIN protocol will occupy more memory than the current version since the data structure definition was more complex, thus create redundant consumption. Compared with LIN 1.3, LIN 2.1 has defined sporadic frames and up to eight-bytes signals. Using diagnostic APIs will take more resources because the control flow of the diagnostic process should be designed to ensure the sequential property of signal transmission. The initial constant data which is stored in ROM will occupy much more memory than the loop control since the entire LIN communication is predictable by the LDF database.

RAM	500 bytes/ 131072 bytes
FLASH	21528 bytes / 536560 bytes

Table 11: Memory consumption of LIN model

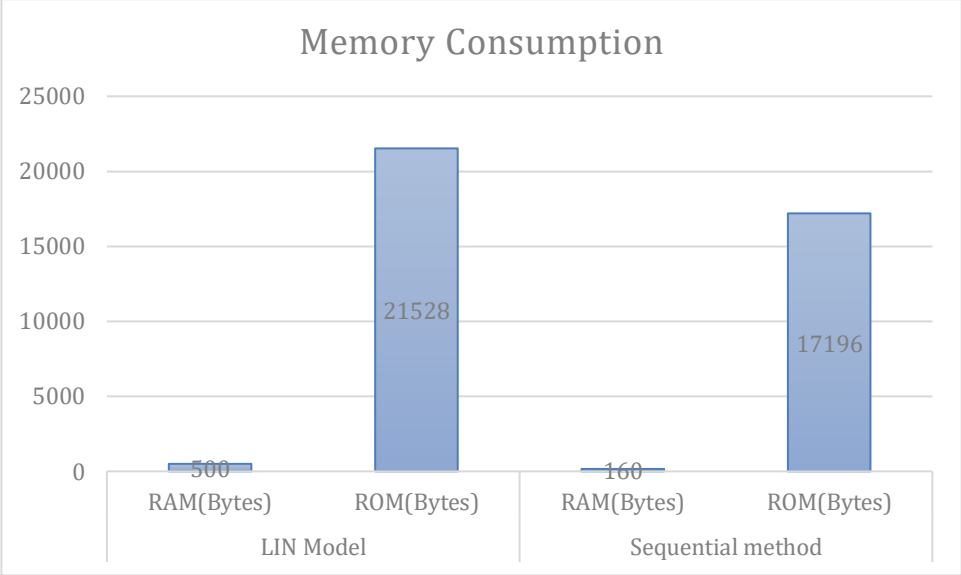

Figure 48: Memory consumption comparison

In conclusion, the LIN model is tested to be feasible in LIN 1.3 and LIN 2.1. Compared with sequential LIN communication, the LIN model can be used in the LIN slave node implementations besides master node implementations with some modification in the diagnostic APIs. The LIN model will utilize more RAM and ROM to store the control flow of the state machines in the layered structure and the predefined constant signal defined in the configuration files generating from the database. Also, the LIN model can separate the LIN master task and the LIN slave task with an identical mechanism, while the sequential method is limited on the master task of a master node.

5.3 Results of LIN node configuration

With the help of a demo configuration tool from ihr GmbH company (Figure 49), the LDF database is converted to a header file including the information formatting in arrays and structure type. The conversion of the frame schedule is shown in Table 12.

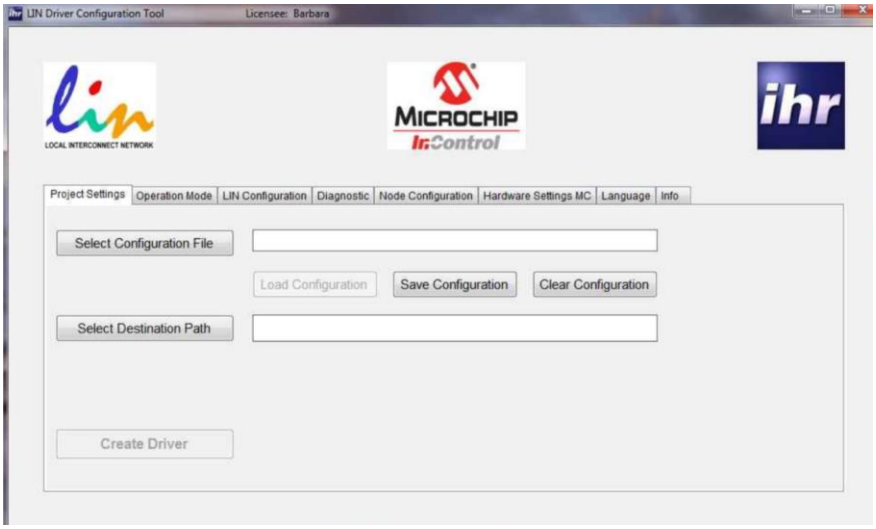

Figure 49: LIN configuration tool

The demo tool will give a simple prototype of each data structure and the specific information from the LDF will be enriched by the users.

LDF database	Header file

Schedule_tables {
Empty{
}
Main {
Demo_Command delay 200 ms;
Demo_Response delay 200 ms;
}
ResponseOnly {
Demo_Response delay 20 ms;
}
CmdOnly {
Demo_Command delay 200 ms;
}
Diag {
MasterReq delay 100 ms;
SlaveResp delay 100 ms;
}
}
	static t_lin_sched_table schedule1[2] =
	{{(l_u16)139,(l_u16)640,(l_u16)2000},\
	
	{(l_u16)76,(l_u16)1152,(l_u16)2000}};
	static t_lin_sched_table schedule2[1] =
	{{(l_u16)76,(l_u16)1152,(l_u16)200}};
	static t_lin_sched_table schedule3[1] =
	{{(l_u16)139,(l_u16)640,(l_u16)2000}};
	static t_lin_sched_table schedule4[2] =
	{{(l_u16)60,(l_u16)2048,(l_u16)1000},\
	
	{(l_u16)125,(l_u16)2048,(l_u16)1000}};
	static t_lin_sched_table schedule5[1] =
	{{(l_u16)60,(l_u16)2048,(l_u16)1000}};
	static t_lin_sched_table schedule6[1] =
	{{(l_u16)125,(l_u16)2048,(l_u16)1000}};
	l_u8 const schedTblSizeList[7] =
	{(l_u8)0,(l_u8)2,(l_u8)1,(l_u8)1,

Table 12: Database configuration schedule table

The configuration header file defines five schedules from the LDF database as well as the timing constraints. The delay between the frame transmissions in the schedule determines the maximum timing interval between two adjacent frames.

The entire process of a signal reception consists of l_cyclic_com_task function that enables the receive state machine, ld_task function that handles the configuration response, and UART receiving interrupt service routines that loads the received data to the buffer. The timing consumption of the functions is described in Table 13, The results are the average timing consumption of 10 identical tests.

TC Nr.	Task	Timing resource
Test case 1	l_cyclic_com_task ()	1480 ns
Test case 2	ld_task ()	590 ns
Test case 3	Uart Rx ISR	1520 ns

Table 13: Timing consumption of functions in LIN model

The communication of one-byte signal will take 3590 ns in total. Since a LIN frame consists up to eleven bytes, it will take up to 39490 ns to finish a frame transaction, within the timing constraints of 20 ms defined in the use case.

5.4 UDS signal exchange process

The signal carrying frame transmission diagram is shown in Figure 50. Compared with the criteria in the sequential method, the break field, synchronization field, PID field, and data field are transferred correctly.

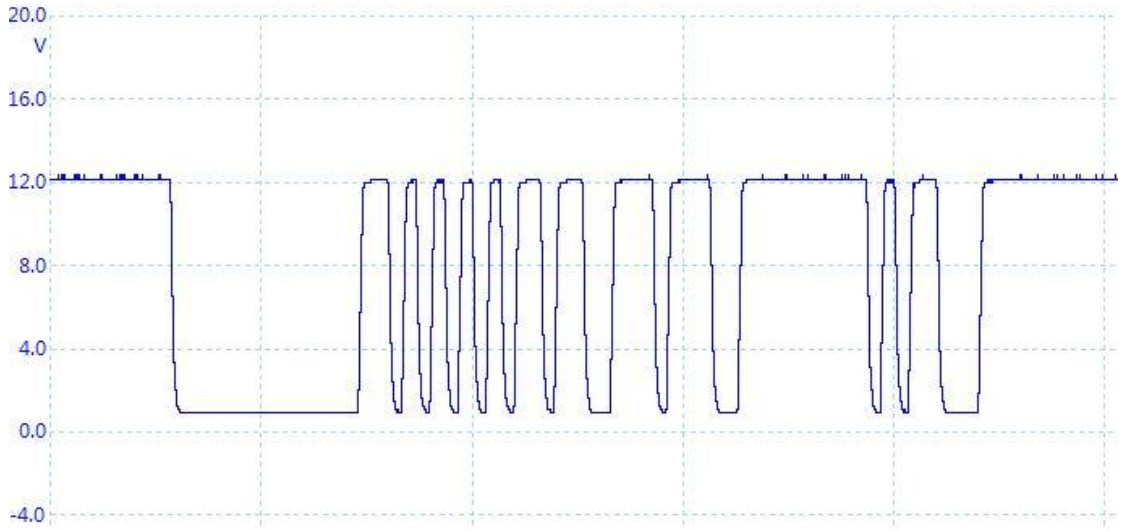

Figure 50: Transmission of signal carrying frames

The break field shall last longer than 13-bit time slot so different baud rates shall be applied in the frame sending. The diagnostic frames with UDS functions transmission are shown in Figure 51 and Figure 52.

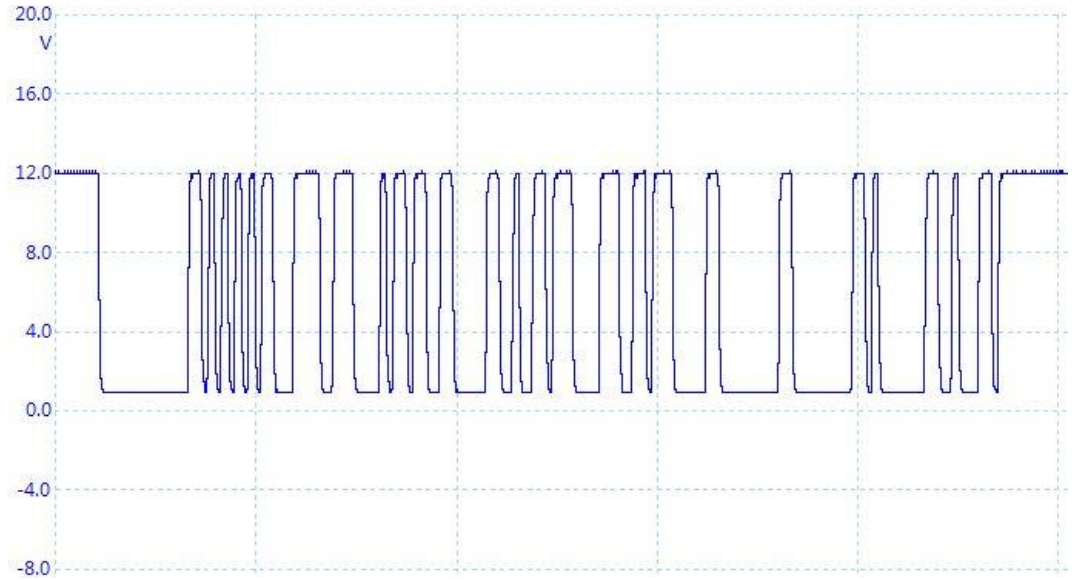

Figure 51: Transmission of master request UDS frame

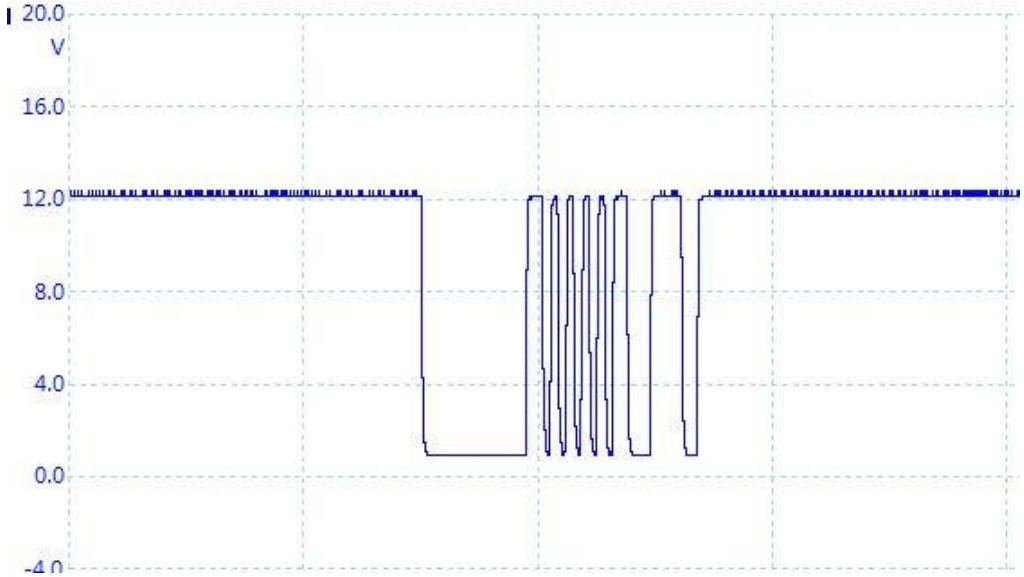

Figure 52: Transmission of the header of slave response frame

The reception of signal carrying frames and diagnostic frames are shown in Figure 53 and Figure 54. The process is detected in the Rx port of the LIN transceiver. After the reception, the extracting data will be loaded to local buffer and local diagnostic receive buffer. Then the application layer shall process and track the following variation of each result. The data should indicate the measurement category defined in ECU instruction.

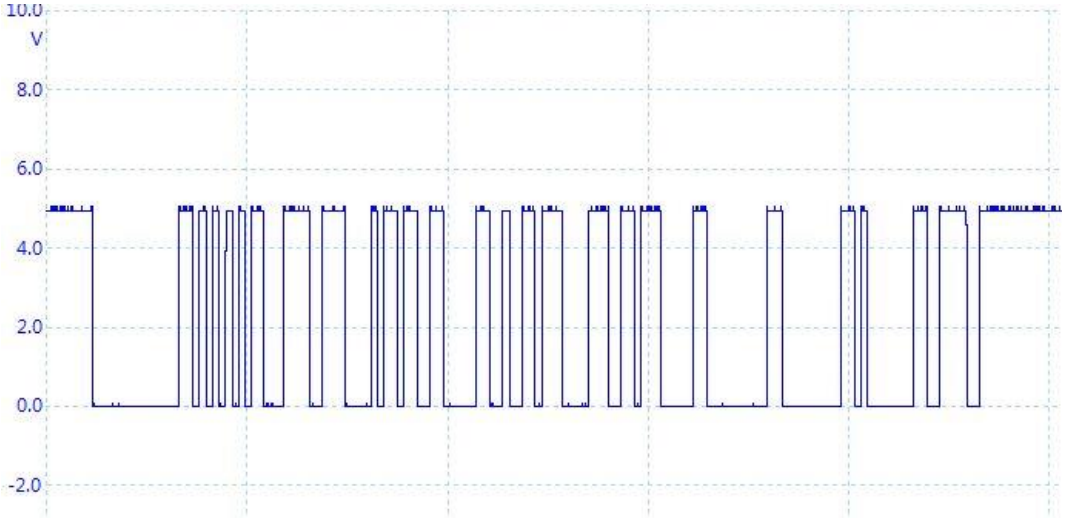

Figure 53: Reception of UDS frame 1

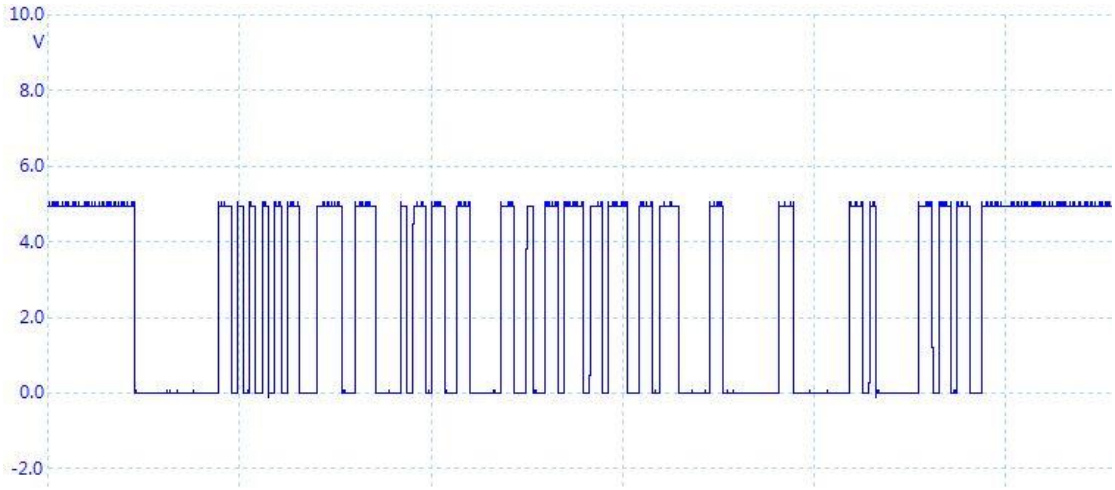

Figure 54: Reception of UDS frame 2

The LIN model is still capable when there are different UDS functions defined in the LDF database. When changing the schedule table and the time base, the configuration process can detect the change by the parser of the configuration tool and reflect in the index in the header file, thus realize the reconfigurability.

In conclusion, the LIN model can provide sufficient UDS functions within the timing constraints defined in the use case's database. The model is also capable of running on the selected hardware platform and the communication process is tested to be reliable by the interaction of layered structures with high flexibility and reconfigurability.

6 Summary and conclusion

This chapter concludes the master thesis. Section 6.1 summarizes the thesis work. Section 6.2 presents the conclusion of the thesis. Section 6.3 specifies some limitations of the project. Section 6.4 introduces some future work.

6.1 Summary

This project successfully develops a new method for the UDS system based on LIN, and the goals in section 1.4 are met. The UDS in the LIN model can be accessed from the database, and the LIN protocol model is able to provide a uniform interface to the LIN network and hide the details of the protocol and message properties from the application. In the project, the knowledge of LIN protocol and UDS standard has been acquired to design and implement an entire LIN communication system from the hardware level to the application level. When it comes to the testing stage, different databases with different UDS have been applied in the LIN model, and the results show great feasibility of the work. From the statistics results, it shows that the method in this project will consume more resources than the sequential method, for some known reasons stated in the specification. The results of timing resource consumption indicate that the UDS system can work within the timing limits in the database. In the related works, the AUTOSAR architecture was raised to implement the LIN communication through a PC in the LIN network, usually with the method of a dedicated software like CANoe or CANalyzer. In this thesis, the AUTOSAR-based LIN model is proved to be feasible on the MCU, and the functions of the LIN network can be modified at the top-level database without exploring the complex LIN protocol.

With the study of standard LIN workflow, the project meets the requirements defined in the LIN specification 2.1, the development process is based on LIN APIs, which are defined in the official documentation. The standardized process will provide convenience for future maintenance, and offer benefit

within the future cooperation with the certified companies by the LIN consortium.

The project also provides a specific set of APIs for the LIN slave node building. Since the idea of a LIN master node is to add the header transmission function on a slave node, the master's and slave's APIs are capable of running on the same node. Also, other functions of unified diagnostic services are supported, such as assigning node addresses and reading node addresses.

6.2 Conclusion

After the implementation process of the LIN model with a UDS use case in the application layer, the following findings can be acquired according to the design process and analysis of testing results.

- The LDF database can be used to describe the behavior of the LIN communication, and the node configuration process can be done within a third-party parser to fill or replace the entities' properties.
- The LIN model inherits the flexibility and reliability of the AUTOSAR structure by the specific design of the basic software layers. The predefined APIs' contents are designed and filled to compose LIN driver, LIN interface, and LIN communication services.
- Compared with the sequential method, the LIN model can provide relatively limited UDS functions with much higher flexibility and reliability, since dependence exists between the layered protocol structure.

6.3 Limitations

The work is based on a specific type of PIC32 MCU, thus the configuration bits and peripheral settings are only applied on the chip. Being different from the initial plan, the development process and the database configuration process cannot run on the hardware platform now, thus decrease the integrity. The limitation is also partly due to the capability of the MCU. Since the MCU can only read the binary machine code, the newly changed database cannot load on the MCU. In fact, the tester cannot run the compilation either.

On the other hand, the UDS services supported by the Blueair Cabin ECU are limited in read-by-id with service ID 0xB2, the data acquisition cannot be performed by other service IDs though the service transmission has been proved to be feasible. The code inside the ECU is invisible, and no programmable port is provided for the program update in ECU. Without building with the LIN protocol model, the LIN slave node in the system does not have the same flexibility as the master node.

Furthermore, the work only implemented the LIN protocol stack to realize the communication service branch in the AUTOSAR basic software layer. Run time environment and system service are not supported to enable the communication between software components inside the LIN node.

6.4 Future work

The demand for smart sensors and actuators has been rising recently with the development of the vehicle industry. As a result, the complexity of automotive in the vehicle bus system witnesses a significant increase. To implement an integrated and reusable protocol model may be the best solution for services update in the LIN system. In the work, the database is a relatively separated part of the stack protocol. Since the LIN network and the node capability are predefined, the application layer replacement can merge the dedicated parser and switching signal's transmission in the database which includes all the services. Few code optimizations are performed in the thesis, so some LIN services use redundant software components in the implementation, and more memory resource has been occupied. Several optimization algorithms can be inserted in the process of compilation and assembly code can be used to replace complex register operations to adapt the work to another hardware platform, which is beneficial for micro LIN systems design in the future.
The AUTOSAR structure we built in this work is limited in the branch of communication services. The system services, I/O services, and complex driver branches will be built in the future work to have a complete AUTOSAR

LIN protocol stack model. The model will enable the possibility of multi-components design for LIN nodes.

On the other hand, the stability and reliability of the LIN model in this thesis is an everlasting topic to be optimized. Motor Industry Software Reliability Association (MISRA) has published the MISRA-C standard to facilitate C code safety, security, portability, and reliability in the context of embedded systems, specifically those systems programmed in C90 and C99. The work may be processed by several MISRA-C checking tools to strengthen the reliability. Also, the LIN model should adapt the content and algorithms in the LIN APIs if the LIN specification gets updates in the future.

References

[1] LIN Consortium Staff, *LIN Specification Package 2.2A,* LIN-SUBBUS Consortium, 2010.

[2] Wei Jing Zhao, Bo Chen, Rui Tao Wang, Xuan Zhang, and Wei Deng. "Design of a vehicle-mounted CO2 measurement model," *Peking Vehicles,* Sep., pp.27-29, 2017.

[3] Francke Thomas and Hammar Magnus, "LIN baserad kommunikationsmodul för GPS," Master thesis (diva2:543041), KTH, Sweden, 2008.

[4] Feng Luo and Yue Yin Xie, "LIN Flash Bootloader Based on UDS," *Journal of Automation and Control Engineering*, vol. 4, no.1, pp. 47-52, 2016.

[5] Ya Hui Shao, "Development of a Window Control System Based on LIN," Master thesis, Chongqing University, China, 2007.

[6] Guang Sheng Han, "Development of the Automobile Door System Based on Technology of LIN Bus," Master thesis, Jilin University, China, 2005.

[7] Hong Su, "Research on Anti Pinch Control System of Power Window Based on LIN Bus," Master thesis, Harbin Institute of Technology, China, 2015.

[8] Xian Cheng Ming and Bo Xu, "Design of an Intelligent Control System for Automotive Lighting Based on LIN," *Modern Electronics Technique*, vol. 14, pp.161-164, 2019.

[9] Ze Fang Shu and Yun Lin, "Design of an Electronic Window System Based on LIN and Network Description File," *Electromechanical Engineering Technique*, vol. 40, pp.59-61, 2011.

[10] Ze Fang Shu, Xiao Shan Peng, Yu Qing Chen, and Hong Wu Zhou, "A Standardized Design of LIN Bus System Based on LDC and Network Description File," Automated Application, vol. 2, pp. 5-7, 2015.

[11] Mary Tamar Tan, Brian Bailey, and Han Lin, "LIN Basics and Implementation of the MCC LIN Stack Library in 8-bit PIC Microcontrollers," [Online]. Available: https://www.microchip.com/wwwAppNotes/AppNotes.aspx?appnote=en585355.

[12] Xiang Yan Li, "Research and realization of a communication model based on AUTOSAR," Master thesis, Dept. Automation, UESTC, China, 2012.

[13] Xiang Yan Li, Yun Li, and Liu Xiang Tang, "Realization of LIN communication based on AUTOSAR," *Computer Engineering,* vol. 38, pp. 208-211, 2012.

[14] Road vehicles - Unified diagnostic services (UDS) – UDS on local interconnect network (UDSonLIN), ISO standard 14229-7, 2015.

[15] Salcianu Monica, and Fosalau Cristian, "A new CAN diagnostic fault simulator based on UDS protocol," in Proc. International Conference and Exposition on Electrical and Power Engineering, Iasi (RO), 2012.

[16] Microchip, "32-bit Microcontrollers," PIC32MX7XX Family Data Sheet, 2016. 09.

[17] Road vehicles – Unified diagnostic services (UDS) – Specification and requirements, ISO standard 14229-1, 2013.

[18] "AUTOSAR Layered Software Architecture," [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards.

[19] "AUTOSAR Specification of Diagnostic Communication Manager," [Online] Available: https://www.autosar.org/.

[20] Afshin Etemad, "Simulation based verification for LIN/CAN application layer," Master thesis, Dept. Signals and Systems, Chalmers University, Sweden, 2016.

[21] J. J. Johan Elegered, "AUTOSAR Communication Stack Implementation," Master thesis, Chalmers University of Technology, Sweden, 2012.

[22] Ross M. Fosler, "Implementing a LIN master Node Driver on a PIC18 Microcontroller with USART," [Online]. Available: https://www.microchip.com/wwwAppNotes/AppNotes.aspx?appnote=en012046.

[23] Jan Dakermandji, “An AUTOSAR diagnostic platform,” Master thesis (diva2:543007), Dept. Machine Design, KTH, Sweden, 2008.

[24] Ya Gang Wang, “Analysis and transplant of embedded bootloader mechanism,” *Computer Engineering*, vol. 36, ch. 6, 2010.

[25] “Specification of PDU Router,” [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/.

[26] “AUTOSAR specification of Module LIN State Manager,” [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards [Accessed: 2017 12.08].

[27] Peng Fei Song, Yang Zhang, Xiang Long Wu and Yu Fan Lan, “Design and Implementation of the Adaptive Control System for Automotive Headlights Based on CAN/LIN Network,” in Third International Conference on Instrumentation, Measurement, Computer, Communication and Control, 2013.

Appendix A

LIN description file database

```
LIN_description_file;
LIN_protocol_version = "2.1";
LIN_language_version = "2.1";
LIN_speed = 19.2 kbps;

Nodes {
        Master: MCU, 5 ms, 1 ms; //time base and jitter
        Slaves: ECU1;
}
Signals {
   Vin :   8, 0, ECU1, MCU;  //supply voltage
   Vout:   8, 0, ECU1, MCU;
   Aout:   8, 0, ECU1, MCU;  //output current
   assignECUStatus: 8, 0, MCU, ECU1;
   assignedECUStatus:8, 0, ECU1, MCU;
   Temp: 8, 0, Ecu1, MCU;
}
Frames {          //unconditional
   changeStatus : MCU, 0x33, 1 {
                        assignECUStatus, 0;
   }
   Status: ECU1, 0x11, 1{
      assignedECUStatus, 0;
   }
}

/**********
Diagnostic_signals {
        Req0_ReadVin: 8, 0; //0x38,identifier
        Req1_ReadVout: 8, 0; //0x39
        Req2_ReadAout: 8, 0; //0x3A
        Req3_ReadTemp: 8, 0; //0x3B
        Resp0_Vin: 8, 0; //Vin
        Resp1_Vout: 8, 0; //Vout
        Resp2_Aout: 8, 0; //Aout
```

```
        Resp3_Temp: 8, 0; //Temp
}
********/
/*
Diagnostic_frames {
   MasterReq: 60 {
      Req0_ReadVin, 0;
      Req1_ReadVout, 8;
      Req2_ReadAout, 16;
      Req3_ReadTemp, 24;
                  MasterReqB4, 32;
                  MasterReqB5, 40;
                  MasterReqB6, 48;
                  MasterReqB7, 56;
        }
        SlaveResp: 61 {
                  SlaveRespB0, 0;
                  SlaveRespB1, 8;
                  SlaveRespB2, 16;
                  SlaveRespB3, 24;
                  SlaveRespB4, 32;
                  SlaveRespB5, 40;
                  SlaveRespB6, 48;
                  SlaveRespB7, 56;
        }
   }
*/
Diagnostic_signals {
        MasterReqB0: 8, 0;
        MasterReqB1: 8, 0;
        MasterReqB2: 8, 0;
        MasterReqB3: 8, 0;
        MasterReqB4: 8, 0;
        MasterReqB5: 8, 0;
        MasterReqB6: 8, 0;
        MasterReqB7: 8, 0;
        SlaveRespB0: 8, 0;
        SlaveRespB1: 8, 0;
        SlaveRespB2: 8, 0;
        SlaveRespB3: 8, 0;
```

```
        SlaveRespB4: 8, 0;
        SlaveRespB5: 8, 0;
        SlaveRespB6: 8, 0;
        SlaveRespB7: 8, 0;
}
Diagnostic_frames {
        MasterReq: 60 {  //0x3c
                MasterReqB0, 0;
                MasterReqB1, 8;
                MasterReqB2, 16;
                MasterReqB3, 24;
                MasterReqB4, 32;
                MasterReqB5, 40;
                MasterReqB6, 48;
                MasterReqB7, 56;
        }
        SlaveResp: 61 { //0x3D
                SlaveRespB0, 0;
                SlaveRespB1, 8;
                SlaveRespB2, 16;
                SlaveRespB3, 24;
                SlaveRespB4, 32;
                SlaveRespB5, 40;
                SlaveRespB6, 48;
                SlaveRespB7, 56;
        }
}

Schedule tables{
        DiagRequestSchedule {
                MasterReq delay 15.000 ms;
        }
        DiagResponseSchedule {
                SlaveResp delay 15.000 ms;
        }
        StatusCheckSchedule {
                changeStatus delay 10.000ms;
                Status delay 10.000ms;
        }
}
```

```
    Node_attributes {
        ECU1 {
            LIN_protocol = "2.2";
            configured_NAD = 0x68;
            initial_NAD = 0x68;
            supplier_id = 0x07;
            response_error = ErrRespECU1;
            configurable_frames {
                AACLin17Fr01;
                CcmLin17Fr10;
            }
        }
}
```

TRITA-EECS-EX-2019: 787