



Waterford Institute of Technology

**PERFORMANCE ANALYSIS OF
AUTOSAR VEHICLE NETWORK
GATEWAYS**

Z h u
朱

W e i D a
韋達

B.Sc. (Hons)

M.Sc.

Supervisor: Brendan Jackman B.Sc., M.Tech.

Submitted to Waterford Institute of Technology
Awards Council, July 2007

ACKNOWLEDGEMENTS

This thesis would have been impossible without the help of the following people.

Firstly, I would like to thank Mr. Brendan Jackman for his encouragement and guidance throughout this project.

I would like to thank Dr. Darren Buttle, ETAS LiveDevices, for his guidance throughout this project.

I would also like to thank the members in our group for their valuable advices and help.

- David Power, Group Supervisor, Department of Computing, Maths & Physics, Waterford Institute of Technology
- Frank Walsh, Group Supervisor, Department of Computing, Maths & Physics, Waterford Institute of Technology
- Kevin Mullery, Group Member, Department of Computing, Maths & Physics, Waterford Institute of Technology.

Finally, I will like to thank my wonderful Parents, and all my friends for encouragement throughout this time.

DECLARATION

Zhu WeiDa

I, **朱 韋達**, declare that this thesis is submitted by me in partial fulfillment of the requirement for the degree M.Sc., is entirely my own work except where otherwise accredited. It has not at any time either whole or in part been submitted for any other educational award.

Signature: _____

Zhu WeiDa

朱 韋達,

May 27th, 2007.

ABSTRACT

Modern vehicles use a variety of data networks to exchange data between their different control modules. These networks operate at different communication speeds to reflect the relative response times of the connected control units. For example, engine control units are connected to a high-speed network while comfort systems such as electric seats are connected to a low speed network. In addition, there are a number of different network operating principles, for example, event-driven and time-triggered. Gateways are required to exchange data between these different vehicle networks. Gateways typically exchange messages between connected networks based solely on the destination and priority of the messages. Such gateways can result in unpredictable message delays depending on the network loading and vehicle operating conditions.

The aim of this research is to develop and evaluate a vehicle network gateway model based on a very general and well designed gateway structure from the automotive industry, which takes into account factors such as the purpose of network messages, vehicle operating conditions, network loading, bus topology and message streams.

The network gateway model is used to evaluate the relative performance of different gateway configurations.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	<i>I</i>
DECLARATION	<i>II</i>
ABSTRACT	<i>III</i>
TABLE OF CONTENTS	<i>IV</i>
TABLE OF FIGURES	<i>VIII</i>
TABLE OF TABLES	<i>X</i>
1. THESIS OVERVIEW	<i>1</i>
1.1 PROBLEM SPECIFICATION	1
1.2 SOLUTION REQUIREMENTS	2
1.3 RESEARCH QUESTIONS	3
1.4 DOCUMENT LAYOUT	4
2. VEHICLE NETWORK PROTOCOLS	<i>6</i>
2.1 INTRODUCTION	6
2.2 EVENT AND TIME TRIGGERED COMMUNICATION SYSTEMS	8
2.3 CONTROLLER AREA NETWORK – CAN	9
2.3.1 CAN AND THE OSI MODEL	9
2.3.2 BUS ARBITRATION	10
2.3.3 CAN BUS ARCHITECTURE	11
2.3.4 MESSAGE AND FRAME FORMATS	12
2.3.4.1 DATA FRAME	12
2.3.4.2 REMOTE FRAME	14
2.3.4.3 ERROR FRAME	14
2.3.4.4 OVERLOAD FRAME	15
2.3.4.5 ERROR DETECTION	15
2.3.5 TIME-TRIGGERED CAN (TTCAN)	18
2.4 LOCAL INTERCONNECT NETWORK – LIN	20
2.4.1 LIN BUS TECHNICAL OVERVIEW	20
2.4.2 LIN PROTOCOL CONCEPT	20
2.4.3 LIN SPECIFICATION	22
2.5 FLEXRAY	24
2.5.1 FLEXRAY ADVANTAGES	24
2.5.2 FLEXRAY APPLICATIONS	27
2.5.3 FLEXRAY PROTOCOL	27
2.5.3.1 FLEXRAY NODE OPERATION	28
2.5.3.2 FRAME FORMAT AND SIGNALS	30

2.6 REFERENCES	32
3. VEHICLE NETWORK DESIGN	33
3.1 INTRODUCTION	33
3.2 NETWORK TOPOLOGIES	34
3.3 NETWORK GATEWAYS	37
3.4 NETWORK DESIGN METHODOLOGIES	39
3.4.1 BASIC NETWORK DESIGN METHODOLOGIES	39
3.4.2 IN-VEHICLE NETWORK DESIGN METHODOLOGIES	40
3.4.3 V MODEL	45
3.4.4 TRADITIONAL DESIGN PROCESS AND ADVANCED DESIGN PROCESS	46
3.5 NETWORK MANAGEMENT	52
3.5.1 OSEK/VDX	52
3.5.2 OSEK/VDX NETWORK MANAGEMENT	53
3.5.3 OSEK/VDX NETWORK MANAGEMENT COMPONENTS	53
3.6 CONCLUSION	55
3.7 REFERENCES	56
4. NETWORK GATEWAY DESIGN	57
4.1 INTRODUCTION	57
4.2 GATEWAY HARDWARE ARCHITECTURES	59
4.2.1 GENERAL GATEWAYS ANALYSIS	59
4.2.2 GATEWAY ARCHITECTURES	60
4.3 GATEWAY SOFTWARE MODULES	63
4.4 OPTIMIZATION OF GATEWAY PERFORMANCE	65
4.4.1 USING OF ROUTERS FOR OPTIMIZING GATEWAY PERFORMANCE	65
4.4.2 HARDWARE/SOFTWARE PARTITIONING AND ROUTING LEVELS	65
4.4.3 ROUTER STRUCTURE IN GATEWAYS	68
4.4.4 DEDICATED HARDWARE FOR ROUTER	70
4.4.5 PROTOCOL ANALYSIS	72
4.5 CONCLUSION	74
4.6 REFERENCES	75
5. SIMULATION	76
5.1 INTRODUCTION	76
5.2 SIMULATION METHODOLOGIES	79
5.3 SIMULATION TECHNOLOGY	80
5.3.1 THE BASIC QUEUING THEORY	80
5.3.2 MARKOV PROCESS AND ITS MAJOR MODELS	82
5.3.2.1 MARKOV PROCESS	82

TABLE OF CONTENTS

5.3.2.2	DISCRETE AND CONTINUOUS MARKOV PROCESS	84
5.3.2.3	EXAMPLES OF QUEUING MODELS	84
5.4	SIMULATION TOOL – MATLAB/SIMULINK AND SIMEVENTS	87
5.4.1	BACKGROUND	87
5.4.2	THE MATLAB SYSTEM	87
5.4.3	SIMULINK AND STATEFLOW	90
5.4.4	SIMEVENTS	90
5.4.4.1	SIMEVENTS QUEUES AND SERVERS	91
5.4.4.2	SIMEVENTS PATHS AND ROUTING TECHNIQUES	92
5.6	CONCLUSION	93
5.7	REFERENCES	94
6.	LITERATURE REVIEW SUMMARY	95
7.	METHODOLOGY	96
7.1	METHODOLOGY OVERVIEW	96
7.2	SIMULATION DESIGN	97
7.2.1	SIMULATION PROCESS	97
7.2.2	SIMULATION PROCESS RELATED TO THIS RESEARCH	100
7.3	SIMULATION TOOL SELECTION	103
7.4	CONCLUSION	105
7.5	REFERENCES	106
8.	GATEWAY MODEL REQUIREMENTS SPECIFICATION	107
8.1	SYSTEM ARCHITECTURE OVERVIEW	107
8.2	AUTOSAR GATEWAY STRUCTURE	108
8.2.1	MESSAGE TRANSMISSION TYPES	110
8.2.2	MESSAGE RECEPTION TYPES	113
8.2.3	AUTOSAR PDU ROUTER	114
8.3	GATEWAY BUFFER REQUIREMENTS	118
8.4	CONCLUSION	119
8.5	REFERENCES	120
9.	SIMULATION MODEL DESIGN AND IMPLEMENTATION	121
9.1	INTRODUCTION	121
9.2	REST BUS SIMULATION MODEL	123
9.2.1	COMMUNICATION BUS SIMULATION MODEL	124
9.2.2	EXPERIMENTAL MEASUREMENT OF MESSAGE TRANSMISSION TIMES	124
9.2.3	BUS LOAD SIMULATION	128
9.2.4	MESSAGE TRANSMISSION TIME SIMULATION	130

TABLE OF CONTENTS

9.2.5	TIME-TRIGGERED BUS SIMULATION MODEL	131
9.3	AUTOSAR GATEWAY SIMULATION MODEL	133
9.3.1	COM MESSAGE GENERATION	134
9.3.2	BUFFER SELECTION	136
9.3.3	MESSAGE ROUTING	137
9.3.4	RATE CONVERSION	138
9.4	TRANSMISSION TYPE MODELLING	140
9.5	RECEPTION TYPE MODELLING	143
9.6	SIMULATION MODEL DESIGN VALIDATION COMMENTS – FROM AN EXPERT IN AUTOMOTIVE INDUSTRY	144
9.7	REFERENCES	145
10.	TESTING	146
10.1	INTRODUCTION	146
10.2	TESTING TOOLS	148
10.2.1	REAL-TIME GRAPHIC REPRESENTATION	148
10.2.2	MATLAB WORKSPACE	149
10.3	VERIFICATION OF GATEWAY SIMULATION MODEL	150
10.4	VERIFICATION TESTING CASES	151
10.5	VALIDATION OF GATEWAY SIMULATION MODEL	159
10.5.1	OBJECT OF VALIDATION	159
10.5.2	VALIDATION PROCEDURE	159
10.6	OPTIMISATION OF GATEWAY PERFORMANCE	164
10.6.1	EVALUATION CASES TABLE	164
10.6.2	EVALUATION CASES	164
10.7	CONCLUSION	181
10.8	REFERENCES	182
11.	CONCLUSIONS	183
11.1	RESEARCH SUMMARY	183
11.2	RESEARCH CONCLUSION	185
11.3	AREAS FOR FURTHER RESEARCH	187
12.	APPENDIX A BIBLIOGRAPHY	188
12	BIBLIOGRAPHY	188
13.	APPENDIX B SAE PAPER	192
13	SAE PAPER	192

TABLE OF FIGURES

Figure 2.1: OSI model and layered architecture of CAN	9
Figure 2.2: CAN bus Arbitration	11
Figure 2.3: CAN network according to ISO 11898	11
Figure 2.4: Data Frame	12
Figure 2.5: Remote Frame	14
Figure 2.6: Error State Diagram of a CAN node	17
Figure 2.7: TTCAN Message Schedule	19
Figure 2.8: LIN bus topology	20
Figure 2.9: LIN bus transmission cycle	21
Figure 2.10: LIN Frame	21
Figure 2.11: LIN Bus Tool Chain	23
Figure 2.12: FlexRay Topologies	24
Figure 2.13: Comparisons of Protocol Data Rates	26
Figure 2.14: FlexRay Node	28
Figure 2.15: FlexRay State Transitions	29
Figure 2.16: FlexRay Error State Transitions	30
Figure 2.17: FlexRay frame format	30
Figure 3.1: Star Topology	35
Figure 3.2: Bus Topology	35
Figure 3.3: Ring Topology	36
Figure 3.4: Central Gateway	38
Figure 3.5: Multi Gateways	38
Figure 3.6: Backbone Gateways	38
Figure 3.7: Design Flow Process	39
Figure 3.8: Network Design Process	41
Figure 3.9: Example of Functional Allocation	43
Figure 3.11: the V Model	46
Figure 3.12: Traditional Network Design Process Effect	49
Figure 3.14: Traditional V Model	50
Figure 3.15: Advanced V Model	51
Figure 3.16: In-vehicle Network Management Environment	53
Figure 3.17: NM equipment with two CAN systems	54
Figure 4.1: The Basic Structure of Fail Silent Units (FSU)	59
Figure 4.2: Single-process Gateway	61
Figure 4.3: Dual-processor with Services Separation	61
Figure 4.4: Dual-processor with Domain Separation	62
Figure 4. 5: Software Components in the system communication via the VFB	64
Figure 4.6: Software Architecture of an AUTOSAR ECU	64
Figure 4.7: Message Definition	66
Figure 4.8: HW/SW partitioning of dual processor gateway	67
Figure 4.9: The Gateway Structure with a Dedicated Hardware	68
Figure 4.10: A Basic Gateway Structure	68
Figure 4.11: Router Management Structure	69
Figure 4.12: Router structure with hardware support	71
Figure 4.13: The Structure of a Hardware Router	72
Figure 4.14: CAN and FlexRay Communication	73
Figure 5.1: The Example of a Discrete System	77
Figure 5.2: The Example of a Continuous System	77

Figure 5.3: Basic Model of Queuing Process	80
Figure 5.4: MIMO States	85
Figure 5.5: MIMO States	86
Figure 5.6: MIMO States	86
Figure 5.7: The Example of the FIFO Queue Block	92
Figure 7.1: Simulation Process	100
Figure 8.1: System Architecture	107
Figure 8.3: Non TP-PDU-TX without Triggered.....	110
Figure 8.4: Non TP-PDU-TX with Trigger Transmit	111
Figure 8.5: TP-PDU-TX	112
Figure 8.6: Non TP-PDU-RX	113
Figure 8.7: TP-PDU-RX	114
Figure 8.8: Non TP-PDU-Gateway without Rate Conversion (Non-Triggered).....	115
Figure 8.9: Non TP-PDU-Gateway without Rate Conversion (Triggered).....	115
Figure 8.10: Non TP-PDU-Gateway with Rate Conversion	116
Figure 8.11: Gateway Block	117
Figure 9.1: In-vehicle Network Simulation Model	122
Figure 9.2: Rest Bus Block	123
Figure 9.3: Single Message Transmission Code	126
Figure 9.4: Formula for Messages per Second	129
Figure 9.5: Message transmission details	129
Figure 9.6: Network Status Display	130
Figure 9.7: Bus Load for 250 Kbits/Sec Simulations	130
Figure 9.8: S-Function Builder for simulation time	131
Figure 9.9: Time Scheduled Messages Transmission	132
Figure 9.10: Gateway Block	133
Figure 9.11: Gateway Simulation Model	133
Figure 9.12: Random Message Generation	134
Figure 9.13: Message Length S-Function	135
Figure 9.14: S-Function Process	136
Figure 9.15: Buffer Selection	136
Figure 9.16: Routing table steps	138
Figure 9.17: Routing Table Block	138
Figure 9.18: S-Function Builder for Rate Convert	139
Figure 9.19: Non Triggered Message Generation	140
Figure 9.20: Non-triggered Transmission (TP or Non-TP)	141
Figure 9.21: Triggered Transmission in Gateway Simulation Model	142
Figure 9.22: Triggered Transmission in Receive Node Simulation Model	142
Figure 9.23: Time Scheduled Messages Reception	143
Figure 10.1: Model Testing	147
Figure 10.2: Routing Information Scope	148
Figure 10.3: Buffer Selection Scope	149
Figure 10.4: Routing Table Workspace	149
Figure 10.5: Gateway Validation Procedures	161

TABLE OF TABLES

Table 2.1: Description of FlexRay Static Segment and Dynamic Segment	25
Table 2.2: Vehicle Network Standard	26
Table 2.3: FlexRay and CAN Comparison	27
Table 3.1: Example of functions in vehicle	42
Table 4.1: In-vehicle Network Protocol differences	72
Table 5.1: Data Types in Details	89
Table 5.2: SimEvents Routing Blocks	92
Table 5.3: SimEvents Routing Techniques	92
Table 9.1: Message Attribute definitions	124
Table 9.2: Function Description	127
Table 9.3: Messages Transmission Parameter Details	128
Table 9.4: Effective Length of CAN message	129
Table 9.5: Time Schedule Message Identifiers and Periods	131
Table 9.6: Attribute Details	134
Table 9.7: S-Function Input/Output	134
Table 9.8: Routing table parameter explanations	137
Table 9.9: Example of a routing table	137
Table 10.1: Verification Testing Case Table Components Description	150
Table 10.2: Bus Configuration RX	161
Table 10.3: Gateway Simulation Model	162
Table 10.4: Bus Configuration TX	162
Table 10.5: Gateway Model Evaluation Results	163
Table 10.6: Initial Case Setup 1	164
Table 10.7: Initial Case Setup 2	168
Table 10.8: Initial Case Setup 3	172
Table 10.9: Initial Case Setup 4	176

PART ONE

THESIS OVERVIEW

1.1 PROBLEM SPECIFICATION

This aim of this research is to investigate how an in-vehicle network gateway works and how to improve the performance of a gateway optimally.

The automotive industry is entering an exciting and challenging time for electronics system designers. Applications such as infotainment, telemetry, safety, and control require the use of several different networking standards. There are a vast array of networking protocols to choose from, each with advantages and disadvantages. No one protocol satisfies the requirements of all automotive applications.

Therefore there is a need to consolidate data from these dispersed networks and feed the relevant processing ECU with the collected data in terms of sending it to its target destination. A gateway is used as a central hub to interconnect and process data from the vehicle's embedded networks. A typical gateway is composed of several automotive networking interfaces such as CAN, Keyword Protocol over CAN, LIN and FlexRay in addition to embedded micro-controllers and peripheral functions.

1.2 SOLUTION REQUIREMENTS

To design an in-vehicle network gateway requires a full understanding of the infrastructure of the whole in-vehicle network, which includes the in-vehicle network protocols, the in-vehicle network management, and the in-vehicle network gateway.

In order to design a reliable gateway, those variations must be accounted for during testing. Even using methodologies such as rapid prototyping, it is difficult to cater for all the critical physical layer permutations that need to be accounted for, therefore simulating a virtual prototype of a gateway is a more effective solution.

1.3 RESEARCH QUESTIONS

The goal of the research is to build a comprehensive gateway simulation model that allows the AUTOSAR gateways performance to be optimised for different networks configurations and bus loads.

The research investigates a number of key questions:

1. Which aspects of an AUTOSAR gateway configuration have impact on gateway performance?
2. Is the Matlab/Simulink and SimEvents a feasible environment to model and simulate the AUTOSAR defined in-vehicle network gateway system?
3. Can a gateway simulation model be used effectively to optimise gateway performance?

1.4 DOCUMENT LAYOUT

The layout of this thesis is as follows:

Chapter 1: Thesis Overview

This chapter introduces the objectives of this research and the required solutions related to this research. The key research questions are also addressed.

Chapter 2: Vehicle Network Protocols

This chapter discusses three commonly used in-vehicle network protocols in automotive industry. The differences between these protocols are also discussed in detail.

Chapter 3: Vehicle Network Design

This chapter introduces the in-vehicle network design process. In this chapter traditional and advanced design processes are compared.

Chapter 4: Network Gateway Design

This chapter discusses the different requirements for in-vehicle network gateway design from two aspects: hardware and software.

Chapter 5: Simulation

This chapter outlines some simulation techniques, which are useful for this research. Also in this chapter, a very powerful simulation tool in the automotive industry is introduced.

Chapter 6: Literature Review Summary

This chapter reviews the existing literature, which has been extensively covered throughout the development of the thesis.

Chapter 7: Methodology

This chapter describes the methodology used to develop and implement the prototype.

Chapter 8: Gateway Model Requirements Specification

This chapter presents the in-vehicle network system specification introduced by AUTOSAR. This chapter uses sequence diagrams to show different communication scenarios. The sequence diagrams are grouped into three sections: PDU Reception, PDU Transmission and PDU Gateway.

Chapter 9: Gateway Simulation Model Design

This chapter describes the in-vehicle network simulation model by using a simulation tool. The simulation model includes: different protocol controllers (time and event triggered), communication bus and gateway.

Chapter Ten: Testing

This chapter describes the test environment and test cases used to test the system and produce the results. In this chapter two testing stages are carried out: Verification and Validation.

Chapter Eleven: Conclusion

This chapter provides a summary of the work conducted in the present research, whilst describing how the objectives were met. The potential for future development is also discussed.

PART TWO

LITERATURE REVIEW

VEHICLE NETWORK PROTOCOLS

2.1 INTRODUCTION

There are more than ten in-vehicle network protocols currently in use in the automotive industry. Generally they are classified into three basic categories based on network speed and functions (Kopetz, 1993):

- **Class A Multiplexing** is used for convenience features (entertainment, audio, trip computer, etc.) and does not require high bandwidth;
- **Class B Multiplexing** is used for general information transfer (instrument cluster, vehicle speed, legislated emissions data, etc.) and requires medium speed;
- **Class C Multiplexing** requires high bandwidth, reliability, and high data integrity (powertrain control, vehicle dynamics, brake by wire, etc.).

CAN is the most widely used of these protocols. The advantages of CAN Bus include high real-time capabilities, operability in a harsh electrical environment, and easy configurability of the overall system. The CAN protocol, which corresponds to the data rates, is used in different automotive control systems. The high-speed data rates of networking controllers are used for some real-time controls such as engine timing and ABS. The low-speed data rates of networking controllers, which make vehicles more comfortable, are for lighting control and air-condition (Etschberger, 2001).

LIN, as a sub-bus of CAN, is an inexpensive serial bus used for distributed body control electronic systems in vehicles. It enables effective communication for smart sensors and actuators, where the bandwidth and versatility of CAN are not required. Typical applications are door control, seats, and climate regulation. LIN bus, offering

fast time to market, flexible design options, low cost and low power consumption, is a cost-effective complement to CAN.

FlexRay is a flexible network communications system, which meets the requirements of high-speed bus systems that are deterministic, fault-tolerant and capable of supporting distributed control systems. FlexRay protocol is designed to meet the key automotive requirements of dependability, availability, flexibility and a high data rate.

2.2 EVENT AND TIME TRIGGERED COMMUNICATION SYSTEMS

As a distributed real time system, an in-vehicle network has two main design approaches: Event triggered and Time triggered the above support the normal data distribution requirements of the vehicle network.

An Event Triggered system controls signals from non-time events occurring outside or inside the system (Kopetz, 1993). In an event triggered system, messages that are event-based are transmitted right after an event and contain information regarding that event. Once an event happens in such systems, the signaling of events from the protocol controller to the micro-controller is made through the use of an interrupt mechanism. During protocol execution, the event triggered system has to make all scheduling and communication decisions dynamically, with the proper functions being executed according to the current event that took place within the network. An event triggered system is responsible for deciding when a message must be sent (Dilger, 1998).

A Time Triggered system has all of its activities initiated by the progression of time. It uses a time division multiple access (TDMA) method to obtain access to the network bus (Kopetz, 1993). This method gives each node a certain amount of time so that they each have specific transmission time in the network. All nodes have a synchronized clock to prevent any node transmitting out of turn, which results in the concept of a shared global time within the system. Therefore, each node in the network decides when a message is to be transmitted according to this global time. All tasks and communication actions within a time triggered system are periodic and state variables are sampled at predefined points in time. A time triggered system is less flexible than an event triggered system, but it is easier to analyze and test (Kopetz, 1993). More details of time triggered system are discussed in the TTCAN Section 2.3.4 in this chapter.

2.3 CONTROLLER AREA NETWORK – CAN

The section discusses the Controller Area Network (CAN) (BOSCH, 1991).

2.3.1 CAN AND THE OSI MODEL

The seven layers of the ISO Open System Interconnection (OSI) (ISO, 1994) model describe most network protocols. Figure 2.1 shows the Data Link Layer and the Physical Layer of the OSI model corresponding to the Control Area Network (CAN) (RICHARDS, 2002).

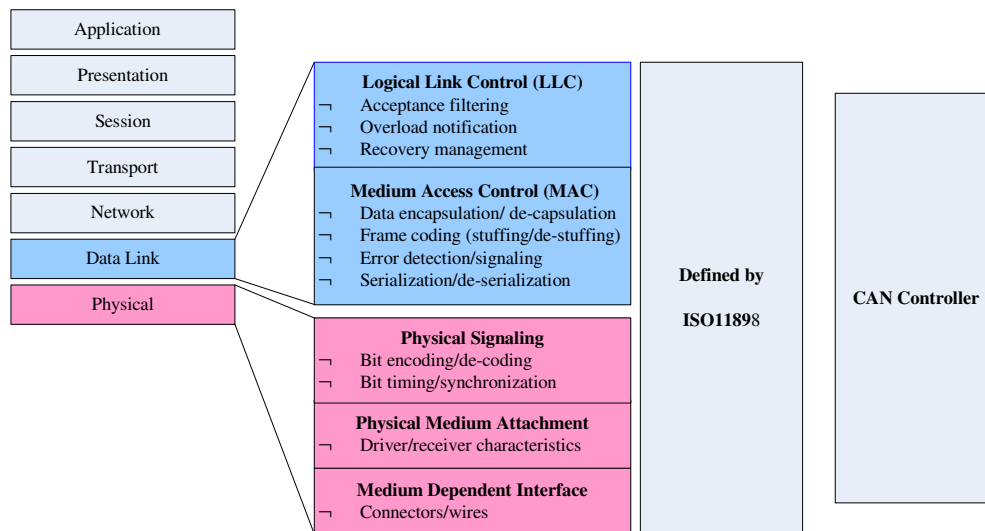


Figure 2.1: OSI model and layered architecture of CAN

CAN has the following features (BOSCH, 1991):

- **Multi-master:** any node can transmit a message to another node when the CAN bus is free.
- **Safety:** CAN provides mechanisms for error detection. The CAN bus error rate is less than 4.7×10^{-11} .
- **Speed and Distance:** when the CAN bus speed is 5KB/s, the furthest distance is 10km; when the CAN bus distance is 40m, the fastest speed is 1MB/s.
- **Arbitration:** if two or more nodes start transmitting messages at the same time the arbitration mechanism is started.

The CAN specification defines the Data Link Layer. The Logical Link Control (LLC) manages the overload control and notification, message filtering and recovery management functions. The Medium Access Control (MAC) performs the data

encapsulation/de-capsulation, error detection and control, bit stuffing/de-stuffing and the serialization and de-serialization functions.

The International Standards Organization (ISO) has defined a standard, which incorporates the CAN specification and the physical layer. The standard, ISO-11898 (ISO, 1993) was originally created for high-speed in-vehicle communications using CAN. ISO-11898 specifies the physical layer to ensure compatibility between CAN transceivers.

2.3.2 BUS ARBITRATION

The CAN protocol uses carrier sense multiple access with a collision avoidance (CSMA/CA) mechanism to arbitrate access to the bus (CIA, 2007). It employs a priority mechanism using numerical identifiers to resolve collisions when two or more nodes want to transmit simultaneously. On the CAN bus a zero represents the dominant bit, which is used to overwrite a one (a recessive bit). Therefore, if there are two nodes, one transmitting a one, another transmitting a zero, the bus results in a zero level.

When two or more nodes want to transmit, they monitor the entire bus to check if there is any bus activity. If there is no activity on the bus, they start to transmit their message identifiers (most significant bit first), yet still monitoring the bus levels. If one node transmits a recessive bit on the bus and another transmits a dominant bit, the bus will result in a dominant level. Therefore, the node transmitting a recessive bit will see a dominant bit on the bus (situation where B loses in Figure 2.2) and stop transmitting any further information. In this situation, the node with the lowest number identifier number gains access to the bus and transmits its message. Any node that has lost during the arbitration process then waits until the bus becomes free before re-transmitting its message.

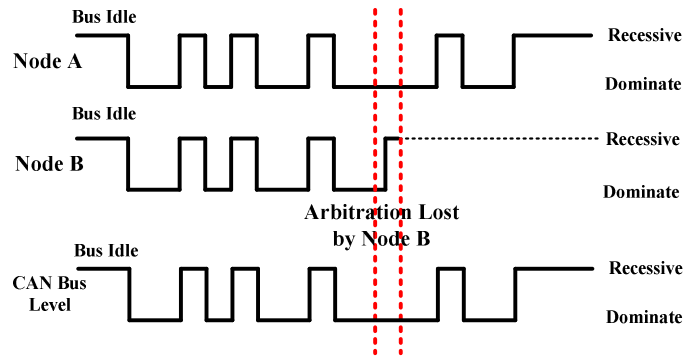


Figure 2.2: CAN bus Arbitration

The CAN protocol uses the bus arbitration mechanism so that the node with the highest priority (lowest value in the identifier field) will continue to transmit without having to back off the bus. It means that CAN has a very predictable behaviour and is very efficient in its use of the bus bandwidth.

2.3.3 CAN BUS ARCHITECTURE

Figure 2.3 shows an example of a typical CAN bus architecture. Although ISO-11898-2 (ISO, 1993) does not specify the mechanical wires and connectors, this specification does require that the wires and connectors meet the electrical specification. In this specification the data rate is defined up to 1 Mbit/s. The high-speed standard specifies a two-wire differential bus whereby the number of nodes is limited by the electrical bus load. The specification also requires 120Ω (nominal) terminating resistors at each end of the bus. The common mode voltage ranges from -2V on CAN_L to +7V on CAN_H. The nominal specific propagation delay of the two-wire bus line is specified at 5ns/m. All these figures are valid only for a 1 Mbit/s transfer rate and a maximum network length of 40m.

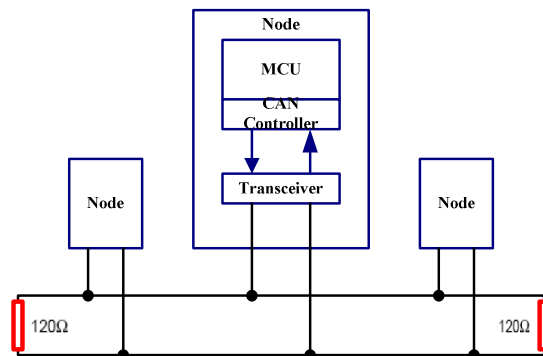


Figure 2.3: CAN network according to ISO 11898

2.3.4 MESSAGE AND FRAME FORMATS

Each CAN message is transferred in the format of a Frame; there are four types of message frames (BOSCH, 1991):

- **Data Frame:** data is transmitted from a transmitter to one or several receivers.
- **Remote Frame:** bus nodes (receivers) can request the transmission of a data frame of the same frame identifier by a source.
- **Error Frame:** signals an error detected by a bus node (transmitter or receiver) and destroys the frame.
- **Overload Frame:** provides for an extra delay between a preceding and a succeeding data or remote request frame to prevent buffer overruns in a receiver.

The original CAN specifications, that is, Versions 1.0, 1.2, and 2.0A define the message identifier as having a length of 11 bits with the possibility of 2048 different message identifiers. However, the updated version, Version 2.0B extends the identifier's length to 29 bits, which means that both 11 and/or 29 bits can be used.

2.3.4.1 DATA FRAME

In CAN bus communication, data is transmitted from a transmitter to one or several receivers.

A data frame consists of seven different fields, which are SOF (Start-of-Frame), Arbitration Field (Identifier and RTR), Control Field, Data Field, CRC Field, Acknowledgement Field and EOF (End-of-Frame). Figure 2.4 shows a data frame.

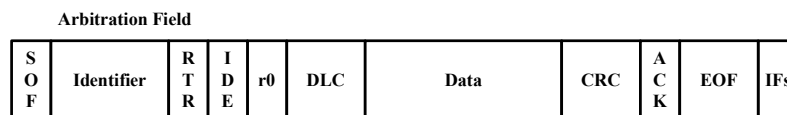


Figure 2.4: Data Frame

Start of Frame (SOF)

This bit is the beginning of a data frame, and is represented by a single dominant bit. A bus node is allowed to start bus arbitration only when the bus is idle, and all the nodes have to be synchronized to the leading edge caused by the start bit of the frame.

Arbitration Field

This field consists of an ID field and a RTR (Remote Transmission Request) bit.

The CAN supports two types of data frame formats: standard frame and extended frame formats. The essential difference between them is the length of the identifier. A CAN standard frame uses 11 bits as the identifier (known as CAN 2.0A), while a CAN extended frame has an identifier with a length of 29 bits (known as CAN 2.0B).

Meanwhile the RTR bit is required to distinguish between the data frame and the remote frame. If a frame is identified as the data frame, the RTR bit takes a dominant value; otherwise the RTR bit is set to recessive. Data frames take precedence over remote frames.

Control Field

This field consists of an Identifier Extension (IDE) bit, which can be used to distinguish between the CAN standard frame and the CAN extended frame. A reserved bit (r0) is defined as a dominant bit. The Data Length Code (DLC) consists of 4 bits, which are used to specify the number of bytes in the Data field.

Data Field

A data field has 0~8 bytes, with each byte consisting of 8 bits.

CRC Field

This field is used to check the integrity of the frame. It consists of a 15 bit CRC sequence for frame checking and a recessively transmitted delimiter bit. The CRC field is calculated by a 15 bit generator polynomial ($x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$) followed by a recessive CRC delimiter bit.

Acknowledgement Field

This field consists of the so-called “Acknowledgement Slot” and a succeeding “Acknowledgement Delimiter” bit. In the Acknowledgement Field, the transmitter sends two recessive bits. Correct messages are acknowledged by the receivers by transmitting a dominant bit in the ACK slot regardless of the result of the acceptance test.

EOF

Every data and remote frame is delimited by a flag sequence of seven recessive bits.

2.3.4.2 REMOTE FRAME

A remote frame is sent by any node to request a message from another node on the network. Compared to the data frame the remote frame has the RTR bit at the recessive state and contains no data field.

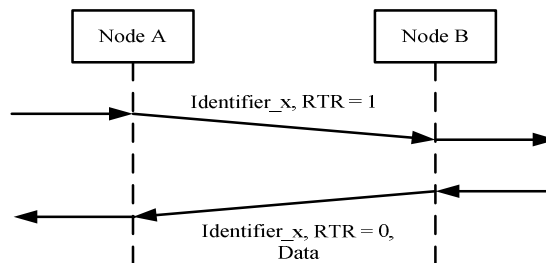


Figure 2.5: Remote Frame

In order to recognise which Data Frame is requested, the initiating node sends a message whose identifier is the same as that of the required Data Frame. Figure 2.5 shows the principle of a data request cycle.

2.3.4.3 ERROR FRAME

Any node detecting a bus error will generate an error frame, which consists of two fields: an Error Flag field and an Error Delimiter field.

The Error Delimiter uses 8 recessive bits to allow the bus nodes to restart bus communications clearly after an error.

The Error Flag (refer to Figure 2.6), of the node detecting the error, decides the content of the Error Flag field. There are two types of error flags: Active Error flag and Passive Error flag.

- If detecting an error in an error state of “error active” on the network, the node will send out an Active Error flag, which is eight dominant bits.
- If detecting an error in error state “error passive” on the network, the node will send out a Passive Error flag, which is eight recessive bits.

2.3.4.4 OVERLOAD FRAME

An overload frame is composed of an overload flag and an overload delimiter. The overload flag consists of a sequence of six consecutive dominant bits. It also destroys the fixed form of the intermission field. As a consequence, all other nodes detect an overload condition, thus they transmit an overload flag.

The overload delimiter consists of eight recessive bits. After transmitting an overload flag every node monitors the bus until it detects a recessive bit. Subsequently every node is finished transmitting its overload flag and all nodes transmit a further seven recessive bits to complete the eight-bit overload delimiter.

2.3.4.5 ERROR DETECTION

The CAN protocol provides the following error detection mechanisms: Bit Check, Frame Check, CRC (Cyclic Redundancy Check), Acknowledgement Check and Stuff Rule Check.

1) Error Types

Bit Check

Every transmitting node monitors whether the bus level transmitted differs from the actual level on the bus. If a transmitted bit value is different from the bit value being monitored, a “bit error” is detected.

The overwriting of a recessively transmitted bit level by a dominant level during the arbitration phase as well as during the ACK slot is not interpreted as a bit error. The overwriting of a passive error flag is also not interpreted as a bit error by a transmitting node.

Frame Check

If a fixed-form bit field contains one or more illegal bits, a “form error” is detected. A receiver monitoring a dominant bit at the last bit of EOF does not interpret this as a form error.

CRC

The CRC sequence composed of the CRC calculation results from the transceiver. If the calculated CRC sequence is different from the sequence received, a “CRC error” is detected.

Acknowledgement Check

If a transmitter determines that a message has not been acknowledged then a ACK Error is flagged.

Stuff Rule Check

Bit stuffing is generally used for various purposes:

- Bringing bit streams that do not necessarily have the same or rationally related bit rates to a common rate, or to fill buffers or frames.
- Synchronizing several channels before multiplexing or to rate-match two single channels to each other.
- Limiting the number of consecutive bits of the same value in the data to be transmitted.

In CAN frames, a bit of opposite polarity is inserted after five consecutive bits of the same polarity. This practice is called bit stuffing, and the “Non Return to Zero” (NRZ) coding is adopted. Therefore, in data exchange, six consecutive bits of the same type (111111 or 000000) are considered an error.

Each node detects a violation of the bit-stuffing rule (“stuffing error”) as soon as it detects the sixth consecutive bit of equal level in a frame field, which shall be coded by the bit stuffing method.

2) Fault Confinement

Each node on the bus, depending on the Transmit and Receive error counter’s values, can be in one of the three states.

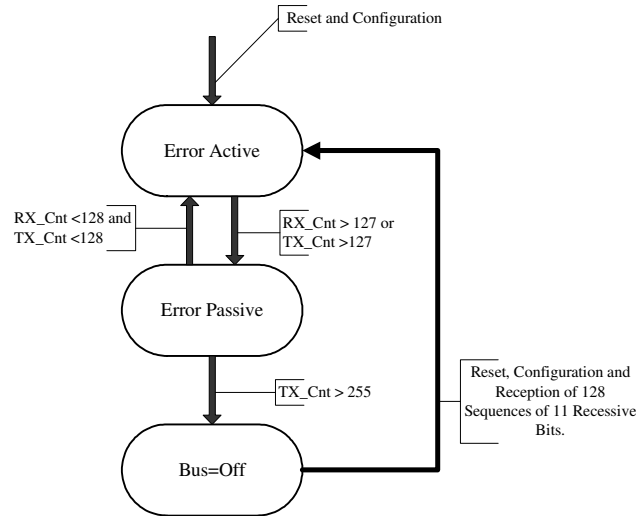


Figure 2.6: Error State Diagram of a CAN node

Error-active

An “error-active” network node takes part in bus communication and sends an active error flag when it detects an error. This is the default node state at reset.

Error-passive

An “error-passive” network node has already accumulated a relatively high transmit or receive error count, thus it has monitored a significantly higher error rate over a longer period of time.

Bus-off

A node in the “bus-off” state is not allowed to have any influence on the bus.

Figure 2.6 shows the error state diagram of a CAN node. After a reset, a node is in the error-active state. If one of the two error counts exceeds the value 127, the monitor demands the MAC sub-layer node enter the error-passive state. A node becomes error

active again when both error counts drop to a below that of 128. A node is disconnected from the bus and is in the bus-off state when the transmit error count exceeds 255. From this state, a node can re-enter the error-active state, with error counts reset to 0, only after reset and reconfiguration and when it has detected 128 sequences of eleven consecutive recessive bits. This measure ensures that a possibly erroneous reset node cannot disturb communication again immediately after reset. Thus up to 128 further frames can be transmitted undisturbed even at a very high busload.

2.3.5 TIME-TRIGGERED CAN (TTCAN)

The communication in the classic CAN network is event triggered; peak loads may occur when the transmission of several messages is requested at the same time (HARTWICH, 2000). CAN uses an arbitration mechanism to ensure that the transmission sequence of all messages corresponds to their identifier priority. However, some mission critical sub-networks within the upcoming generations of vehicle systems, e.g., the x-by-wire system, will require more deterministic behaviour during the communication. For example, when the bus load is at maximum all the safety related messages must still be transmitted. Therefore, it must be possible to allocate a specific amount of time to the message for a high priority transmission. TTCAN is one way to solve the issue.

The time manager in TTCAN communication transmits reference messages regularly. Each individual message to be transmitted is given the time slots in a sequence of time windows following the reference message. There are three types of time windows: Exclusive Time Windows, Arbitrating Time Windows and Free Time Windows.

- Exclusive Time Windows are used for periodic messages.
- Arbitrating Time Windows are used for event messages.
- Free Time Windows are used for future extensions of the network.

The sequence of the time windows is called a basic cycle, which starts with a reference message and contains an off-line configured set of time windows.

Figure 2.7 shows an example of a TTCAN message schedule (Thomas Fuhrer).

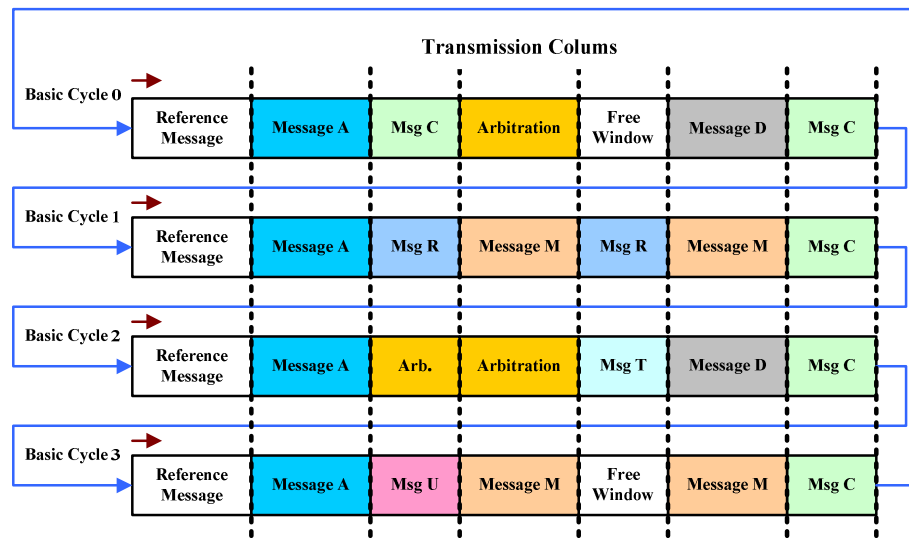


Figure 2.7: TTCAN Message Schedule

Basic Cycle is the time between two reference frames. It is not always identical in order to be able to transmit messages at different periodic frequencies. Several basic cycles are repeated in the vehicle network system unless the vehicle is turned off.

Only one node can send a frame during the exclusive window within a basic cycle. There are also free windows and arbitrating windows, for which the nodes compete for bus access, just as in the regular CAN communication. Several arbitrating windows can be merged. The end of an arbitrating window is always predictable.

2.4 LOCAL INTERCONNECT NETWORK – LIN

2.4.1 LIN BUS TECHNICAL OVERVIEW

LIN (Local Interconnect Network) (LIN—CONSORTIUM, 2003) is a new low cost serial communication system. The communication is based on the SCI (UART) data format, a single-master/multiple-slave concept, a single-wire 12V bus and clock synchronization for nodes without a stabilized time base. Usually, a LIN bus is used in some integration equipment, such as doors, steering wheel, seats and air-conditioning. In these devices, LIN enables a cost-effective communication for smart sensors and actuators. Furthermore, by using digital interface encoding instead of analog interface encoding, electronic devices can be easily connected to in-vehicle network systems and implemented for different diagnosis and maintenance functions including system reprogramming and updating.

2.4.2 LIN PROTOCOL CONCEPT

This section highlights the LIN protocol concept (LIN—CONSORTIUM, 2003)

Basic Operation

LIN bus operation is based on a single-master/multiple-slave concept, as shown in Figure 2.8.

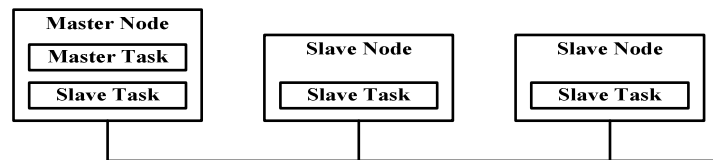


Figure 2.8: LIN bus topology

Each LIN node is divided into two individual parts:

- Master Task, which decides the frame sequence using a schedule table.
- Slave Task, which is responsible for data transfer on the bus, and to allow waking up the entire slave nodes from sleep state.

A typical master node includes a master task and a slave task, but each slave node contains a slave task.

Frame Format

The data unit transferred by a LIN bus is called a data frame. Each frame consists of two parts, as shown in Figure 2.9:

- Message header provided by master task,
- Message response managed by slave task.

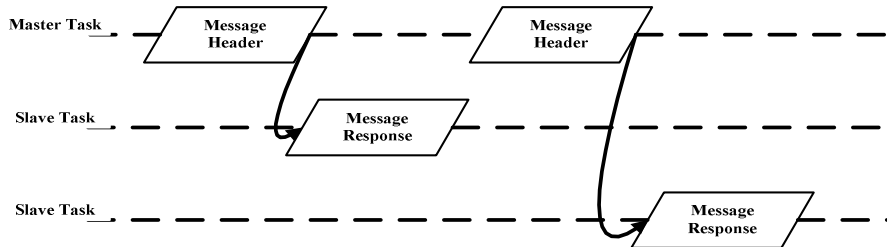


Figure 2.9: LIN bus transmission cycle

Each message header comprises the following parts:

- The synchronization break – at least 13 dominant bits, sent out by the master task, and included in every LIN frame.
- The synchronization byte -- hex value 0x55, which used to synchronize with the master's clock.
- The message identifier – defines unique message content (but not the node address) for receiver side.

The message response of each LIN frame is supplied by a slave task, and it can be divided into Data Field and Checksum, as shown in Figure 2.10:

- Data Field – transfers 1~8 bytes of data.
- Checksum – computed as inverted eight bit sum with carry.

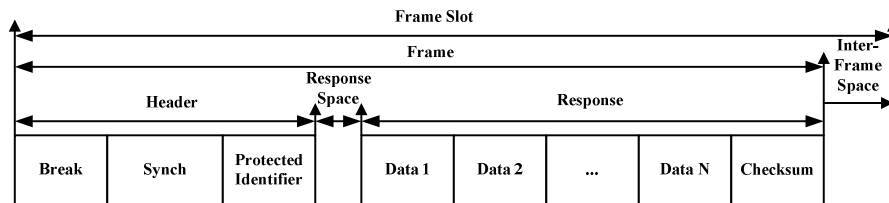


Figure 2.10: LIN Frame

The data content transferred by message responses in the LIN frame consists of three types:

- Signal – the data field of each response includes bits matrix. Under this situation, a frame ID must be between 0x00 and 0x3B.

- Diagnostic information – used to transfer diagnostic and configuration data. This information is 8 bytes long, and has a reserved ID number. ID 0x3C is used by a master request, and 0x3D is used by a slave response.
- Reserved information – used for user defined extension, or for the protocol extension. For example, 0x62 is reserved for user addition, and 0x63 is reserved for future improvements of the protocol.

The message headers transferred by a master task are based on a master schedule table. This schedule table defines the transfer sequence of frames and the interval time between frames. The concept of the schedule table is essentially a mechanism which avoids network overload and guarantees data transfer latency.

2.4.3 LIN SPECIFICATION

The LIN Standard encompasses the specification (LIN-CONSORTIUM) of the transmission protocol, the transmission medium, the interface between development tools and the interfaces for software programming. LIN guarantees the interoperability of network nodes from the viewpoint of hardware and software and a predictable EMC (Electromagnetic Compatibility) behavior.

The Specification consists of three main parts:

The **LIN Protocol Specification** describes the Physical Layer and the Data Link Layer of LIN. The main features of the LIN bus are:

- Single master/multiple slave concept (no bus arbitration),
- 0 to 8 byte message frame,
- Data checksum security and error detection,
- Minimum cost for semiconductor components and
- Guarantee of latency times for signal transmission.

The **LIN Configuration Language Description** describes the format of the LIN configuration file, which is used to configure the complete network and serves as a

common interface between the OEM and suppliers of the different network nodes, as well as an input to the development and analysis tools.

The **LIN API** describes the interface between the network and the Application Program. This concept allows the implementation of a seamless chain of design and development tools and enhances the speed of development and the reliability of the network.

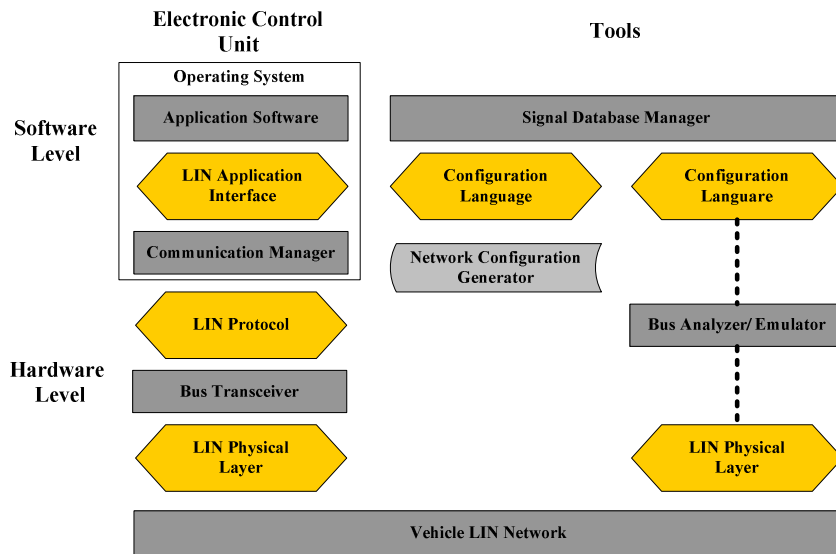


Figure 2.11: LIN Bus Tool Chain

2.5 FLEXRAY

The FlexRay protocol (FLEXRAY-CONSORTIUM, 2000) is the current standard for similar products, which support faster and more highly reliable in-vehicle networks, thus it will lead the whole vehicle electronic control products development for the next generation. FlexRay is the latest in-vehicle network protocol after CAN and LIN, which provides more efficient management for the functionalities of safety and comfort, such as the “X-by-Wire.”

FlexRay is a registered trademark of Daimler Chrysler Automotive Group. The FlexRay Consortium promotes the standardization of FlexRay as the next-generation in-car communication protocol.

2.5.1 FLEXRAY ADVANTAGES

FlexRay (FUJITSU, 2006) focuses on some core requirements including high bit rate, channel redundancy flexible data communication and comprehensive topology, as shown in Figure 2.12.

FlexRay is a type of next-generation in-vehicle network protocol which provide high-reliability and high-speed controls. The CAN network has a speed performance limitation of 1 Mbps, while with a maximum data rate of 10 Mbps available on two channels, giving a gross data rate of up to 20 Mbit/sec, FlexRay can potentially offer 20 times higher bandwidth than a CAN when used in similar applications.

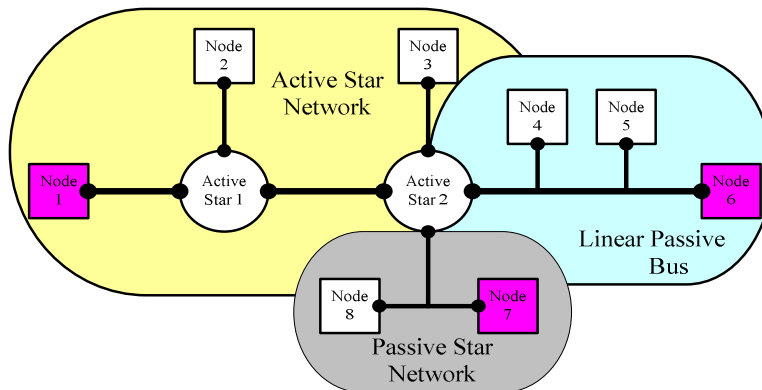


Figure 2.12: FlexRay Topologies

FlexRay has many reliability features that are not available in CAN. For example, a redundant communication capability enables fully duplicated network configurations and schedule monitoring by hardware. Also, FlexRay has flexible configurations, with support for topologies such as bus, star and hybrid types, as shown in Figure 2.12. Designers can configure distributed systems by combining two or more of these topologies. The topologies of FlexRay will be discussed in the next chapter.

In addition, FlexRay allows both synchronous and asynchronous data transfer to meet the demand for various vehicle systems. For example, a distributed control system usually requires synchronous data transmission.

FlexRay provides both static and dynamic communication segments within each communication cycle. The static segment is configured with the fixed time trigger method and the dynamic segment is configured with the flexible event trigger method. Table 2.1 below describes each segment.

	Features	Slot length /data length	Priorities	Bus Guardian (BG)
Static Segment	Sends and receives messages with time triggers	Fixed length	Fixed by fixed TDMA	Because the timing for sending is fixed, it is protected by BG
Dynamic Segment	Sends and receives messages with event triggers	Variable length	Transmission in ascending order of ID	Because the timing of sending is undetermined, it cannot be protected by BG.

Table 2.1: Description of FlexRay Static Segment and Dynamic Segment

In Table 2.1, a concept called Bus Guardian is applied in FlexRay. The bus guardian (BG) in FlexRay manages the schedules and data independently from the communication controller. The bus guardian monitors timing independently. If a gap in timing is found it sends a signal to prohibit the bus driver from transmitting in order to protect the bus status. Simultaneously, it notifies the host of the error. More details of the bus guardian will be discussed in chapter four “Network Gateway Design.”

In addition to operating as a single-channel system similar to CAN and LIN, FlexRay can operate as a dual-channel system. The dual-channel option makes data available via a redundant network – a vital capability for a high-reliability system.

Table 2.2 below shows FlexRay's characteristics meeting real-time control functions.

Class	Communication	Applications	LIN	CAN	FlexRay
A	10K to 125Kbps	Lamps, lights, door locks, power seats, etc.	↕		
B	125K to 1Mbps	Electronic indicators, driving information, automatic air conditioner, etc.		↕	
C	1M to 10Mbps	Engine control, ABS, transmission control, break control, etc.			↕

Table 2.2: Vehicle Network Standard

Figure 2.13 shows the comparison of networking standards by node cost and data rate.

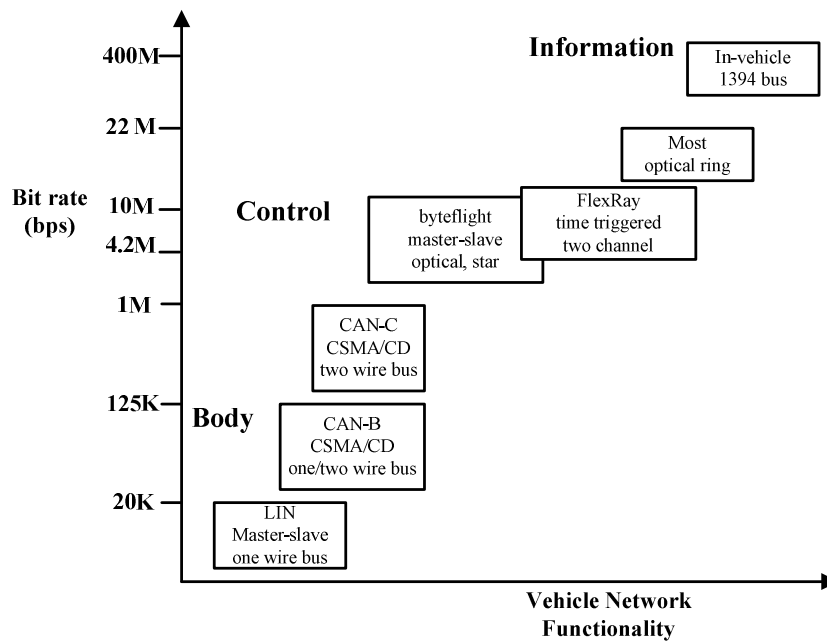


Figure 2.13: Comparisons of Protocol Data Rates

Table 2.3 gives a detailed comparison of FlexRay and CAN.

No.	Item	CAN	FlexRay
1	Baud rate	1 Mbps	10 Mbps
2	Number of channel for one node	1 ch	2/1 ch (optional)
3	Network topology	Bus type	Mix. Of bus and star type
4	Connection node (max.)	16 nodes at 500 kbps	22 nodes (bus) 22/64 nodes (star) 64 nodes (mixed)
5	Physical layer	Metal	Metal/POF
6	Communication	Event triggered	Time triggered, event triggered
7	ID	11/29 bits	11 bits
8	Data length code (DLC)	8 bytes	254 bytes
9	Frame	Data frame, remote frame, error frame, overload frame	Data frame
10	Error status transition	Error active, error passive, bus off (software restoration possible)	Normal active, normal passive, halt
11	Error counter	Status transition counter value fixed	Any status transition counter value
12	Types of errors	Bit error, stuffing error, CRC error, framing error, ACK error	Clock sync. error
13	oscillator	Ceramic and/or crystal	Crystal oscillator (BG separated from CC clock)
14	Network management	software	Hardware (controlled by BD and BG)
15	Bus length	40meters at 1 Mbps	22 meters (in an active star, and between active star)

Table 2.3: FlexRay and CAN Comparison

2.5.2 FLEXRAY APPLICATIONS

FlexRay was developed for X-by-Wire operation in vehicle networks. Suitable FlexRay applications are:

- EPS (Electronic Power Steering) – specifically uses ECUs, it is based on Steering-by-Wire technology.
- ABS (Anti-lock Brake System) – includes VSC (Vehicle Stability Control) and VSA (Vehicle Stability Assist), it is based on Safe-by-Wire technology.
- AT (Automatic Transmission) – replaces existing mechanical control systems with computerized fuel injector, computerized variable intake control system, and computerized idling control system. It is based on Drive-by-Wire technology.

2.5.3 FLEXRAY PROTOCOL

This section highlights the FlexRay protocol (FLEXRAY-CONSORTIUM, 2000).

2.5.3.1 FLEXRAY NODE OPERATION

Each FlexRay node consists of a controller part and a driver part (FUJITSU, 2006), as shown in Figure 2.14.

- Controller: includes a host processor and a communication controller.
- Driver: includes bus drivers and optional bus guardians. The bus driver connects the communication controller to the bus, and the bus guardian monitors access to the bus. The host informs the bus guardian which time slots the communication controller is allocated. The bus guardian then allows the communication controller to transmit data only within these time slots thus enabling the bus driver. If the bus guardian detects a gap in the timing it disconnects the communication channel.

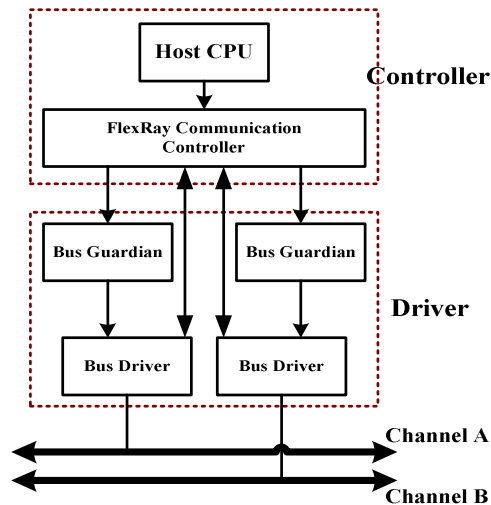


Figure 2.14: FlexRay Node

Status Transition

The nodes of FlexRay have the following basic states, as shown in figure 2.15, which vary from the initial setting to normal communication.

- Configuration State (Default config/config) – used for different kinds of initial setting, including communication cycle and baud rate.
- Ready State – used for internal communication setting.
- Wakeup State – used for waking up the nodes that are not communicating. Under this state, a node sends out a wakeup signal, wakes up and enables the communication controller, bus driver and bus guardians.
- Startup State – used for starting up the clock synchronization and is ready for

communication.

- Normal State – used for indicating the communication is ready.
- Halt State – used for indicating the interrupt of a communication.

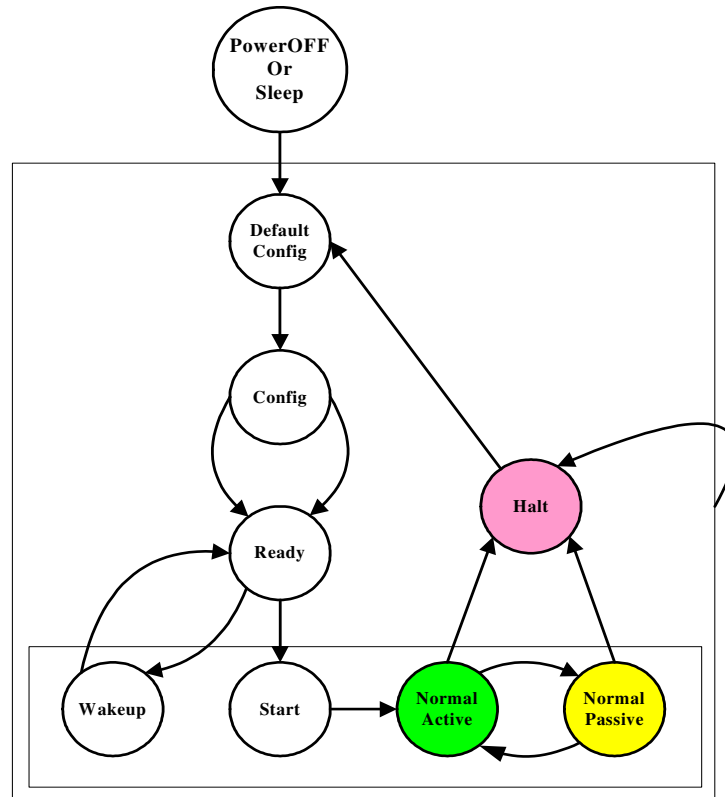


Figure 2.15: FlexRay State Transitions

Error State Transition

The nodes of FlexRay also have an error processing status transition. These status transitions are managed based on an error counter, whose values include a clock synchronization error and clock correction value error depending on the application. When the clock of a node is different to the FlexRay clock synchronisation, a clock synchronisation error is detected. Each FlexRay network has one or more synchronized nodes that transfer synchronisation information. When receiving any synchronisation information, a node will compare its clock with a synchronisation nodes clock and make any necessary changes according to the synchronisation requirements.

Each FlexRay node performs an error count, which counts the number of clock synchronization errors. Nodes monitor the errors related to frame receive and transmit

statuses, which consists of the syntax error, content error, bus error and transfer conflict error. Once a signal node detects one of the errors, it informs the Host CPU. The use of an error counter depends on the applications and system design.

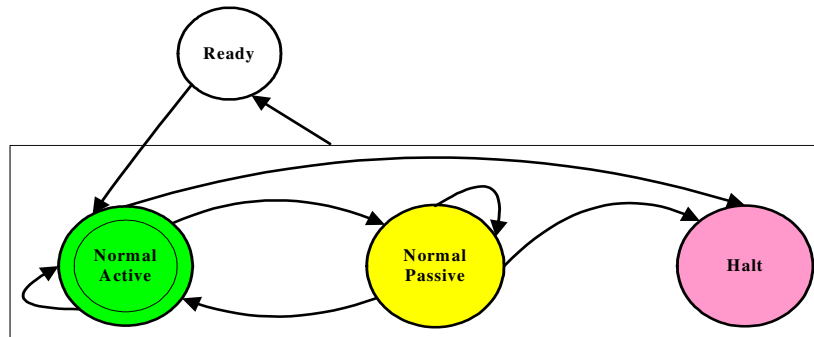


Figure 2.16: FlexRay Error State Transitions

2.5.3.2 FRAME FORMAT AND SIGNALS

Each communication frame of FlexRay consists of three frame segments: Header Segment, Payload Segment and Trailer Segment, as shown in Figure 2.17.

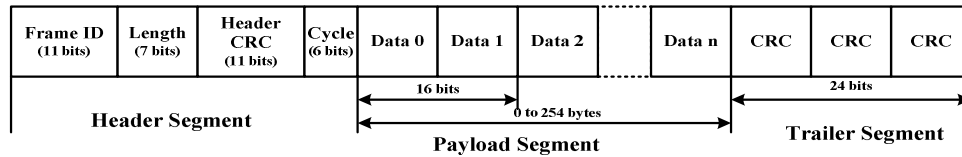


Figure 2.17: FlexRay frame format

Header Segment includes the following bits:

- Reserved bit – used for further expansion.
- Payload preamble indicator – indicates the existence of vector information in the payload segment of the frame. In a static frame, this bit indicates NWVector; in a dynamic frame, this bit indicates message ID.
- Null frame indicator – indicates whether or not the data frame in the payload segment is NULL.
- Sync frame indicator – indicates the existence of a synchronisation frame.
- Start-up frame indicator – indicates whether or not the node-sending frame is the start-up node.
- Frame ID – assigned to each node at system design.
- Length – specifies the data length of the payload segment part.
- Header CRC – specifies the CRC calculation values of Sync Frame

Indicator, start-up Frame Indicator, Frame ID and Length that are calculated by the host.

- Cycle – indicates the cycle count of the node that transfers the frame during the frame transfer time.

Payload Segment includes three parts:

- Data – valid range is from 0 to 254 bytes.
- Message ID – uses the first two bytes of the payload segment for definition, and it can be used as the filterable data on the receiving side.
- NWVector – the network management vector length must be from 0 to 12 bytes and common to all nodes.

Trailer Segment includes the CRC value calculated and specified by hardware. It changes the seed value on the connected channel to prevent incorrect connections.

2.6 REFERENCES

- BOSCH (1991)** CAN Specification Version 2.0. Stuttgart, Robert Bosch GmbH.
- CIA (2007)** Controller Area Network (CAN) -- Protocol. ed., CAN in Automation.
- E.DILGER, T. F., B.MULLER (1998)** The X-By-Wire Concept: Time Triggered Information Exchange and Fail Silence Support by new System Services. SAE.
- ETSCHBERGER, K. (2001)** Controller Area Network. ed. Weingarten, Germany, IXXAT Press.
- F.HARTWICH, B. M., T. FUHRER AND R. HUGEL (2000)** CAN Network with Time Triggered Communication. CIA (CAN in Automation). Robert Bosch GmbH.
- FLEXRAY-CONSORTIUM (2000)** FlexRay Specification Version 2.1. FlexRay-consortium.
- FUJITSU (2006)** Next Generation Car Network -- FlexRay. ed., Fujitsu Microelectronics (Shanghai) Co., Ltd.
- H. KOPETZ, G. G. (1993)** TTP – A Time Triggered Protocol for Fault-Tolerant Real Time Systems. Proc. 23rd IEEE International Symposium on Fault Tolerant Computing. IEEE Press (pg 524-532).
- ISO (1993)** Road vehicle – Interchange of digital information – Controller Area Network. Huting Verlag, Heidelberg, Germany, International Organization for Standardization.
- ISO (1994)** Information Technology – Open System Interconnection – Basic Reference Model – Conventions for the definition of OSI services. Huting Verlag, Heidelberg, Germany, International Organization for Standardization.
- LIN-CONSORTIUM** LIN Concept. LIN-consortium.
- LIN-CONSORTIUM (2003)** LIN Specification Version 2.0. LIN-consortium.
- RICHARDS, P. (2002)** A CAN Physical Layer Discussion. ed., Microchip Technology Inc.
- THOMAS FUHRER, B. M.** Time Triggered Communication on CAN (Time Triggered CAN -- TTCAN). CIA (CAN in Automation).

VEHICLE NETWORK DESIGN

3.1 INTRODUCTION

Today's vehicles contain hundreds of circuits, sensors and many other electrical components. Communication is needed in many circuits and functions of the vehicle. In-vehicle networking is a method for transferring data in distributed electronic modules via a serial data bus, such as CAN, LIN and FlexRay. Applying a serial data bus reduces the number of wires by combining the signals on a single wire through time division multiplexing. Information is sent to individual control modules that control each function, such as anti-lock braking, turn signals, and dashboard displays. (NAVET, 2005)

As the electrical equipment of today's vehicles continues to increase, the need for networking is critical. For example, some high-end luxury cars contain more than three miles and nearly 200 pounds of wiring. The resulting number of connectors creates a reliability nightmare. (NAVET, 2005)

3.2 NETWORK TOPOLOGIES

A network topology structure is a physical layout, which connects different nodes of a communication network. For an in-vehicle network, there are various methods of connection between different protocol nodes, but only few methods can work properly in vehicle networks.

Typical Network Topologies

The most important topologies adopted in vehicle networks are Star, Bus and Ring topologies.

Star Network Topology (BROWN, 1996-2000)

The star network, as shown in Figure 3.1, basically works on a central processor unit, to which all nodes on the network are directly connected. Therefore this central processor unit controls all the information transmission on the network.

Due to its physical structure, a star network has the following advantages and disadvantages:

Advantages:

- Each node has its own connection to the central node,
- Simple integration of further nodes and
- Easy to implement with optical transmission media.

Disadvantages:

- Generally high total length of all connections if nodes are ordered as a geographical line,
- The central node requires N interfaces for the connection of N nodes,
- Communication between nodes only possible via central node and
- Communication is no longer possible if the central node fails.

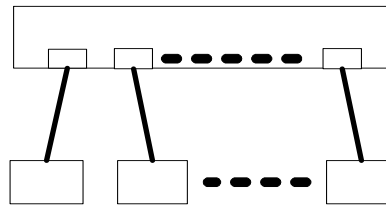


Figure 3.1: Star Topology

Bus Network Topology (BROWN, 1996-2000)

The bus network, as shown in Figure 3.2, is derived from computer bus access control. It connects all electrically passive nodes to a carrier transfer bus. This topology is also called a “diffusion network,” because the data transmitted by one node is available to all nodes. A bus network has the following advantages and disadvantages:

Advantages:

- Lower cabling costs for applications with nodes geographically ordered as a line,
- Simple connecting of a node,
- Simple extension by further nodes without interruption of the operation,
- Failure or de-activation of one node does not affect the other nodes and
- Arbitrary logical communication structures possible.

Disadvantages:

- Limited bus length and number of nodes if signal regeneration by repeaters is not applied,
- Generally, for an electrical bus medium both ends of the bus line have to be terminated by a terminating resistor,
- The implementation of a bus structure with optical media is complicated by the fact that suitable optical branches are still difficult to implement,
- Node identification required.

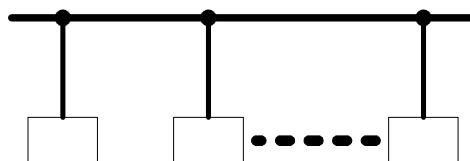


Figure 3.2: Bus Topology

Ring Topology (Brown, 1996-2000)

A ring topology, as shown in Figure 3.3, is implemented by a closed chain of addressed point-to-point connections.

Advantages:

- Implementation of extended networks possible because every node provides signal regeneration.
- Suited for the use of optical transmission media due to the applied point-to-point transmission between nodes.
- Simple identification of nodes possible according to the geographic position of the nodes in the ring.

Disadvantages:

- Total system fails when one of the nodes fails. Therefore additional measures are generally taken, e.g. possibility to bridge the failed node or provision of a redundant ring.
- Operation has to be interrupted for the integration of a new node or replacement of a node.

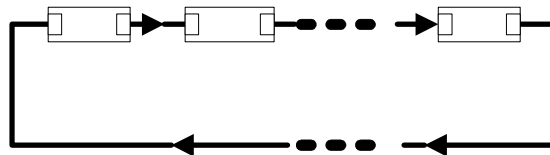


Figure 3.3: Ring Topology

3.3 NETWORK GATEWAYS

A network gateway is a computer system or device that provides communication between different networks. A gateway is a translator when it uses different network protocols, data formats or languages, even with two systems with different system configurations. A gateway repacks the information received, so that this information can meet system requirements. Meanwhile, a network gateway supplies the functions of filtering and safety. Depending on their implementation, network gateways can operate at the application layer of the OSI model.

Gateways become a critical factor in vehicle network design and applications such as infotainment, telemetry, safety and control which require the use of several networking standards. There are a vast array of networking protocols to choose from – each with advantages and disadvantages. No one protocol satisfies the requirements of all automotive applications. There is a need to consolidate data from these networks and perform processing in a central location. As such, a gateway is used as a central hub to interconnect and process data from a vehicle's embedded networks. A gateway is composed of several automotive networking interfaces such as CAN, LIN and FlexRay, in addition to embedded micro controllers and peripheral functions.

General In-vehicle Gateway Types

There are three types of gateway used in vehicle networks: Central Gateway (Super Gateway), Multi Gateway and Backbone Gateway. (EASIS, 2005b)

Central Gateways (Super Gateways): provide a single central gateway for different network systems communicating, as shown in Figure 3.4. It is a very sophisticated gateway, which supports very complex functionalities such as connecting all the different bus systems (High-speed bus and low-speed bus), routing information and converting different message formats.

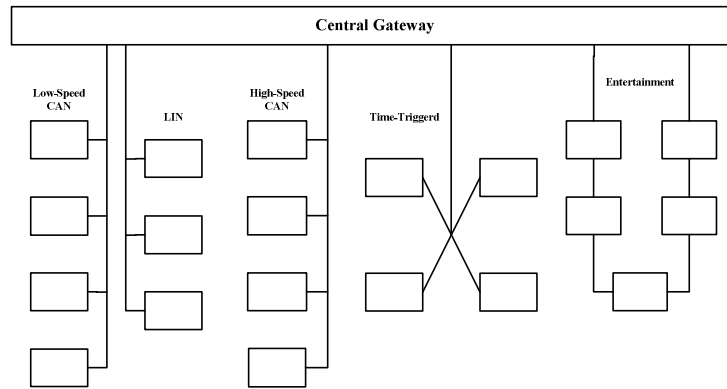


Figure 3.4: Central Gateway

Multi Gateways: are integrated into a single ECU node with the same functionalities as central gateways, as shown in Figure 3.5. It means all communications occurs between ECUs.

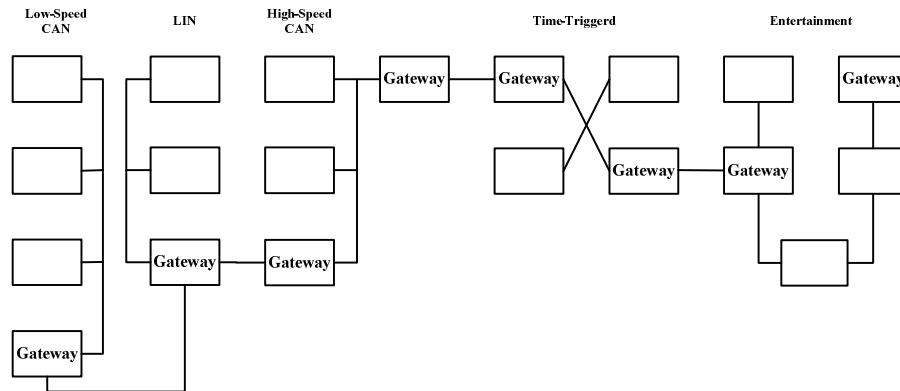


Figure 3.5: Multi Gateways

Backbone Gateways: performs as a front door for each vehicle sub-network of the whole vehicle network, as shown in Figure 3.6. A high-bandwidth backbone (FlexRay) is required to connect those different sub-networks.

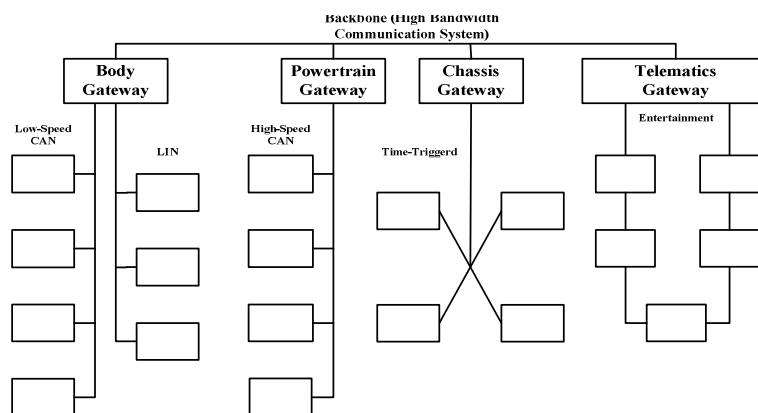


Figure 3.6: Backbone Gateways

3.4 NETWORK DESIGN METHODOLOGIES

3.4.1 BASIC NETWORK DESIGN METHODOLOGIES

In general, each system or network is unique and requires different solutions and methods for design. There are three basic methods (INS, 2002) which can be used to design network architectures and system solutions: the Scientific Method, the Discovery Method, or the Requirements-driven Method. A fourth approach that meets all of the general design and architectural requirements is a combination of the three basic methods, and is referred to as the Tried and True Method (INS, 2002).

A general flow process, as shown in Figure 3.7, can be followed for all approaches of designing network architectures and system solutions. This process begins with a requirement, followed by an analysis of the existing or new technologies and concludes with a solution model or suggested solution. Each of these steps is found in most methods of design when devising a network or system. These steps can be used to define different phases of network design and implementation.

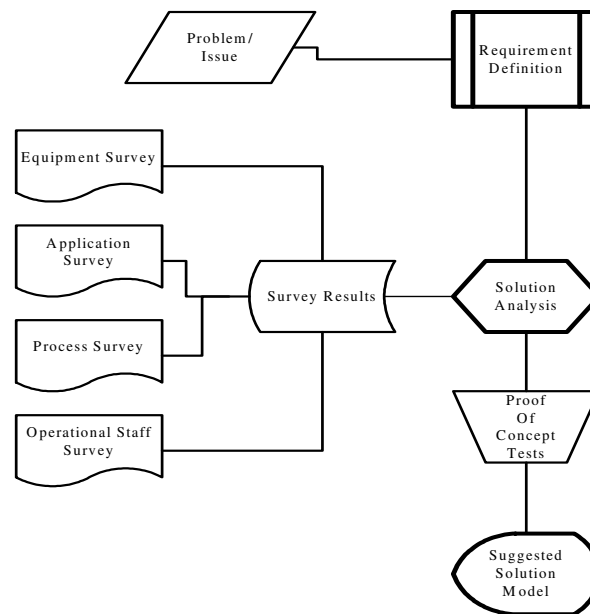


Figure 3.7: Design Flow Process

Scientific Method

The Scientific Method is the same as that method used for physical and theoretical scientific efforts. The process flow begins with the formation of a hypothesis,

followed by testing of the hypothesis, then revising the hypothesis based on experimentation, which results in a theory or stated fact. This method can be applied to a network or systems design using the same order of processes.

Discovery Method

The Discovery Method is a very client-intensive and inclusive process. Generally, the client has an initial understanding of the problem or issue to be solved, but does not have the detailed knowledge to select or design the proper solution. This method is best when the client expects to be completely involved in a team situation.

Requirements Driven Method

The Requirements Driven Method is best applied in two scenarios, for an existing network with problems or a new network. This method is optimally applied by breaking the requirements into component parts, developing and testing individual solutions before they are amalgamated into a network.

Tried and True Method

The Tried and True Method for designing networks and systems draws from all three of the previously mentioned methods. This design method uses five components or phases: defining requirements; surveying existing systems and applications; reviewing “proven” products; testing equipment and application proof of concept; and developing an integrated design and tests. The above description conforms to the ‘V’ cycle methodology employed in most designs found today in the automotive industry. The designer develops a high-level solution based on the requirements as well as a review of the technologies available.

3.4.2 IN-VEHICLE NETWORK DESIGN METHODOLOGIES

“Last minute changes, difficult verification, testing, and similar issues” (HEURUNG) add to the challenges of in-vehicle network design. So, for better, cheaper, and easier

network design it is essential that a canonical and structured engineering design process should be applied.

The Design Process (BROWN, 2006)

The Tried and True Method described earlier in Basic Network Design Methodologies can be used for an in-vehicle network design. Six components or phases are covered by the Design Process illustrated in Figure 3.8.

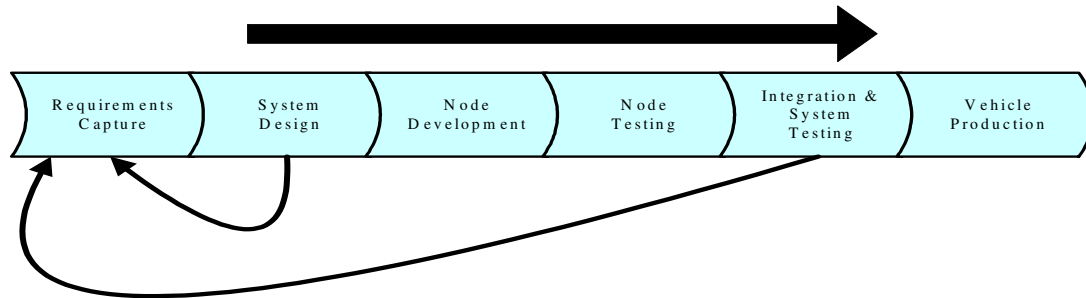
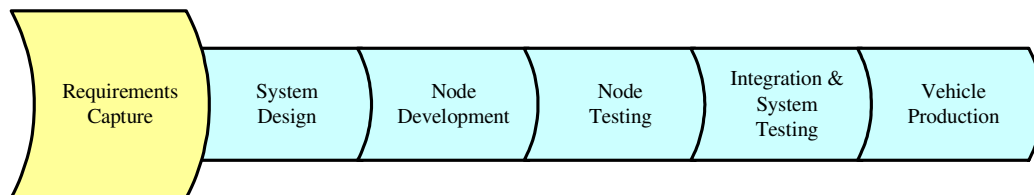


Figure 3.8: Network Design Process

1. Requirements Capture



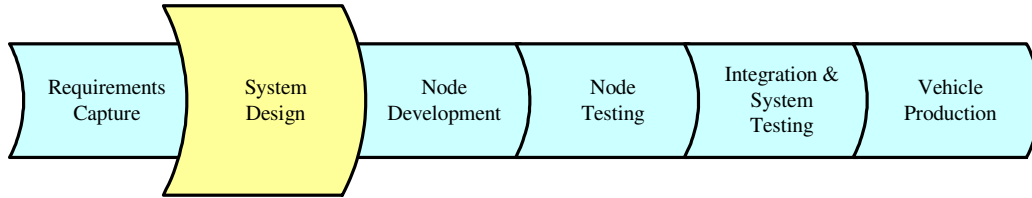
The Requirements Capture stage focuses on defining and analyzing the functions and systems in a vehicle.

Table 3.1 describes different functions in vehicles with their platform versions and variants. For example, in platform version of XJ540ASIA the function of anti-lock braking is a standard configuration. Moreover, the collision avoidance, airbags, 5 speed Trans and auto lighting are option configurations only. But for the platform version X540US, anti-lock braking, airbags and auto lighting are standard configuration, and the collision avoidance and 5 speed Trans are optional configurations. Therefore, the system engineers have to capture all these system requirements, so that these requirements will be considered during the later steps.

Function	Platform Versions & Variants			
	XJ540ASIA	X540US	X540AUS	MN280STD
Anti-Lock Braking	Standard	Standard	Option	NA
Collision Avoidance	Option	Option	Option	NA
Airbags	Option	Standard	Option	NA
5 Speed Trans.	Option	Option	Option	NA
Auto Lighting	Option	Standard	Standard	NA

Table 3.1: Example of functions in vehicle

2. System Design



The System Design step is a multi-step process, which includes four factors for the whole vehicle system: Network Architecture, Functional Allocation and Partitioning, I/O Control and Timing Behaviour.

Network Architecture

As discussed previously, there are many in-vehicle network protocols, and the ECUs with their sensors and actuators are gathered and the data buses are connected to their corresponding protocols (bus architecture, number and position of gateways, etc.), which can directly influence its working efficiency. For example, in a central gateway network, the architecture can be regarded as a bus topology. Low-speed, high-speed and time-triggered buses are connected to the gateway.

Functional Allocation & Partitioning

This sub-step of the system design defines the mapping of hardware and software systems.

Figure 3.9 shows how a typical function is connected to the environment, and how it is structured internally (AXELSSON, 1999).

- Mode Selection: operates in different modes, which could be selected by operators such as switches.
- Control: executes according to the current selected mode. They attempt to control the physical process using the actuators, and base their decision on

sensor values. The user gives the reference value, for example, turning a knob.

- **Diagnosis:** classifies the input signals, and gives information to the mode selector if an error has been found.

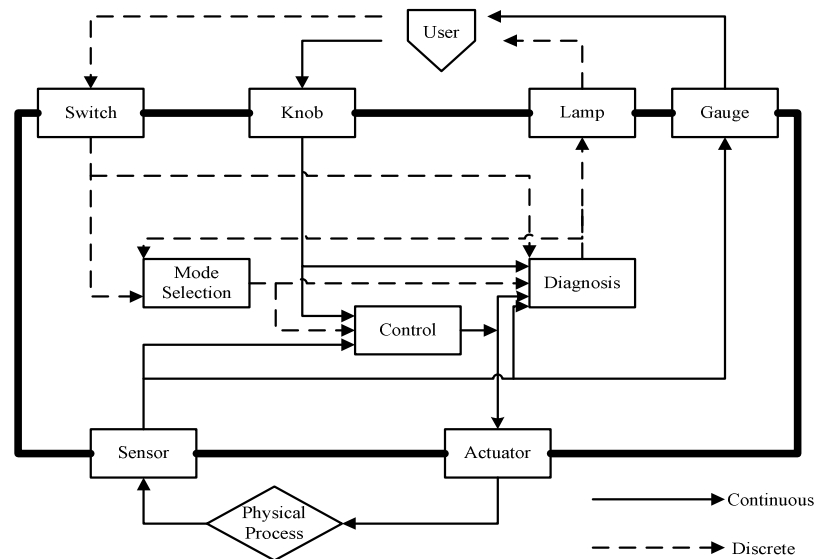


Figure 3.9: Example of Functional Allocation

Figure 3.10 shows how a function can be implemented by two ECUs with the network communication in between (AXELSSON, 1999). The same function described in the functional layer is partitioned in the software layer, for example, parts of it are allocated to the ECUs and the necessary communication is decided. The ECUs implement tasks and data, and the communication transmits signals onto the network. In the support layer, the operating system and device drivers provide the interface between the application software and hardware. On the network, this corresponds to the actual bus frames that are used to transport the application signals. Finally, the hardware layer includes the CPU, memory, sensors and actuators connected to the ECU, as well as the physical network with its associated communication interfaces.

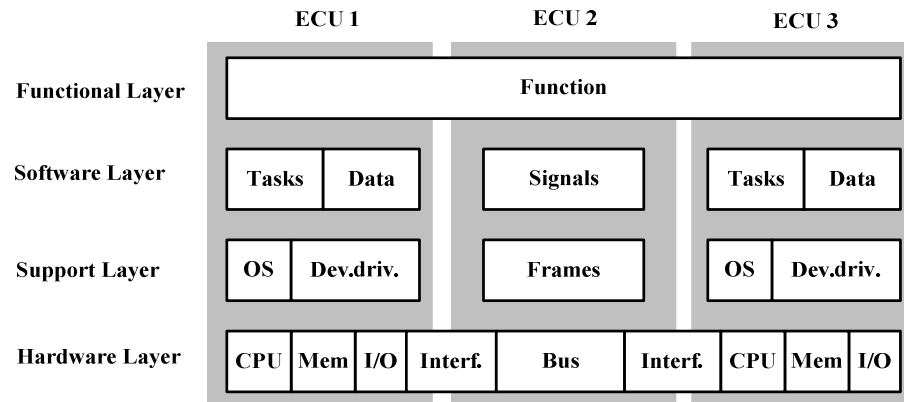


Figure 3.10: Example of Functional Partitioning

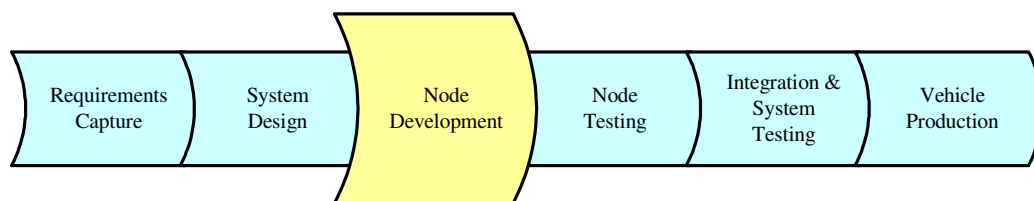
I/O Control and Timing Behaviour

I/O control can be understood as the system behaviour. Behaviour modelling tools such as Simulink can model the I/O interfacing by using certain algorithms they provide themselves. For example, the I/O interfacing in Simulink can be either FIFO or Priority algorithms.

Timing behaviour of the system depends on the prioritization of the ECU operating system tasks, to which the software is distributed and the number and prioritization of bus messages set up for the transmission of signals.

3. Node Development

This step requires the system engineer to generate the application code corresponding to every signal or frame changing in an ECU node.

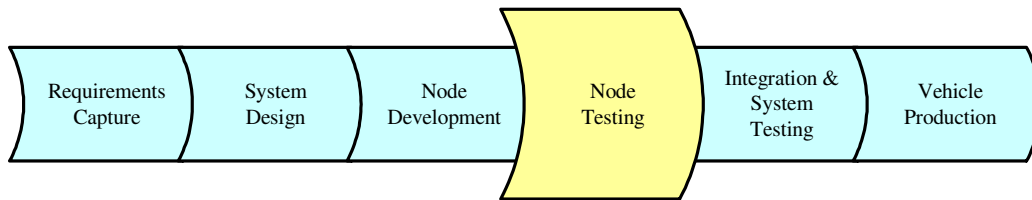


4. Node Testing

This step focuses on performance to ensure each node satisfies the specific requirements. These requirements include:

- Power up Conditions

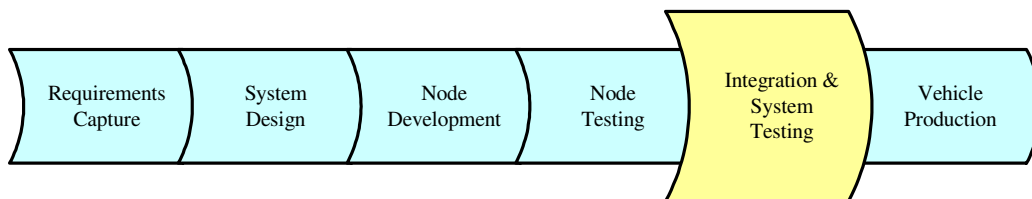
- Startup Sequences
- Error Recovery
- Functional Requirements
- Communication Requirements



5. Integration & System Testing

This step focuses on checking the entire functionality of the in-vehicle network, integrating and connecting every function implemented in the software into a total software system. In this testing environment all the network problems will be identified, which include:

- Requirements Capture
- System Design Process
- Validation of System Design
- Complexity of Network Design
- Feature Content Level
- Testing Environment



6. Vehicle Production

This step is required to make decisions regarding production at a certain stage after testing.

3.4.3 V MODEL

The V model, as shown in Figure 3.11, is one of the most common design processes used in the automotive industry. Paul Rook originally proposed it in late 1980s to

improve the efficiency and effect of the software development. The V model reflects the relationship between testing actions and analyzing actions.

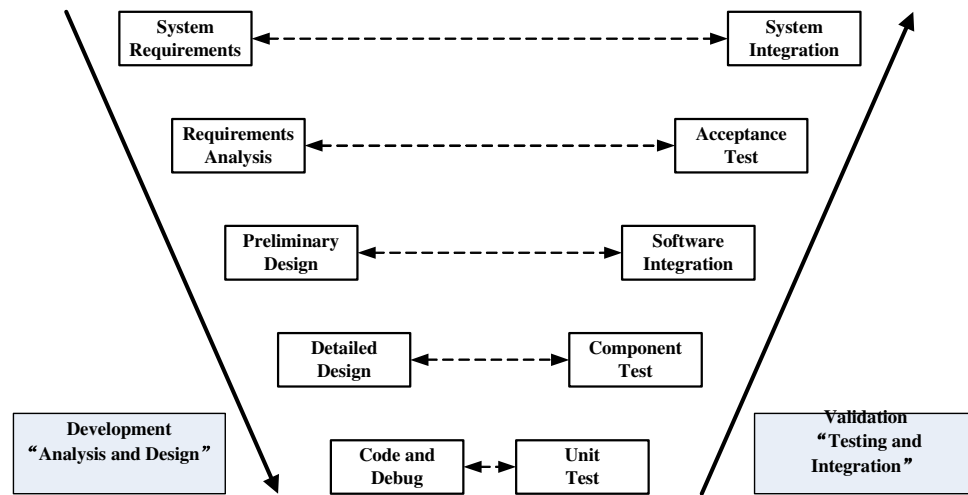


Figure 3.11: the V Model

The left side to right side of this model it represents the basic development process and testing behaviours, and clearly indicates different levels in a testing process and the relations between these testing phases and development process. Additionally, the arrows represent the timing direction, the left side going down representing each phase of the development process, and the right side going up representing each phase of the testing process.

The V model has a limitation; it utilizes the testing process only as a phase after Requirements Analysis, Preliminary Design, Detailed Design and Coding. It is very easy to misinterpret testing as the last phase, since this phase focuses on searching for errors in the program. However, the problems hidden in Requirements Analysis could be found in the later stage of Acceptance Testing.

3.4.4 TRADITIONAL DESIGN PROCESS AND ADVANCED DESIGN PROCESS

Traditionally, when the interaction with other vehicles was low, suppliers were able to apply the electrical system engineering concepts to isolated ECUs, one by one and to adopt this engineering alone and without the need to consider the whole system. However, since multiplexing has become more popular, the interactions between ECUs and subsystems have increased, resulting in substantially more complex

systems and design processes. The differences between Traditional and Advanced design (HEURUNG) focuses on Requirements Analysis, System Design, and Node Development in the Design Process. More detailed differences in the above steps are discussed below.

Requirements Capture between Traditional and Advanced

Traditional Requirements, which include signals grouped to generate a signal database (size, range, priority, and timing) and timing, are mostly based at node level rather than on the whole system and its functionalities. Signal requirements of this step are traditionally are grouped to generate a signal database. However, timing requirements and signals published and subscribed are analyzed for the whole system and its functionalities in advance.

System Design between Traditional and Advanced

Traditionally, signals are manually grouped together and packed into frames based on transmit requirements and priority of data. Signal interactions are also checked manually to verify the system's functional requirements. In traditional system design, network utilization and delay calculations are manually made to make a preliminary assessment of anticipated bus behaviour.

Conversely, advanced system design adopts a network design tool, which provides efficient methods for design a complete system communication. These efficient methods include:

- Collecting all necessary network parameters to allow validation of network concepts prior to starting development.
- Calculating bus loading automatically to confirm network design, which meets all timing requirements and has deterministic communication behaviour.

Node Development between Traditional and Advanced

Traditionally, node development can be started after the initial database describing all the nodes information is distributed. In node development, if the developer makes changes for a signal or frame, they must refer back to the application source code.

Additionally, the software written under traditional node development may not operate the same as the system design level predicated.

In advanced node development, however, network parameter files, which replaced initial databases, distribute node information. In node development, a developer usually develops software to describe the system network. Differing from that of the traditional method, a code generator is used to create the application source code, which means each node signal or frame is individually coded into its object code, so that the developer changes the code without opening the whole system code. Thus the advanced node development provides reconfiguration flexibility and automatic gateway configuration.

Network Design Process between Traditional and Advanced

In a traditional design process (WESTMANN, 2006), the effort, as shown in Figure 3.12, focuses more on integration and testing, not on the definition of the specification. The traditional design process is also called “correct by test,” because many problems are not discovered until the parts are put together, which makes the effort of fixing implementation faults or even design faults much higher.

The disadvantages of the traditional design process include:

- The communication validation is not adequate.
- Real worst-case latencies are non-deterministic.
- Limited bus utilisation.
- Software communication requirements are not completely defined by the design process.
- System reconfiguration is not flexible.
- Time is utilized fixing design during integration.
- Warranty issues are inherently increased.

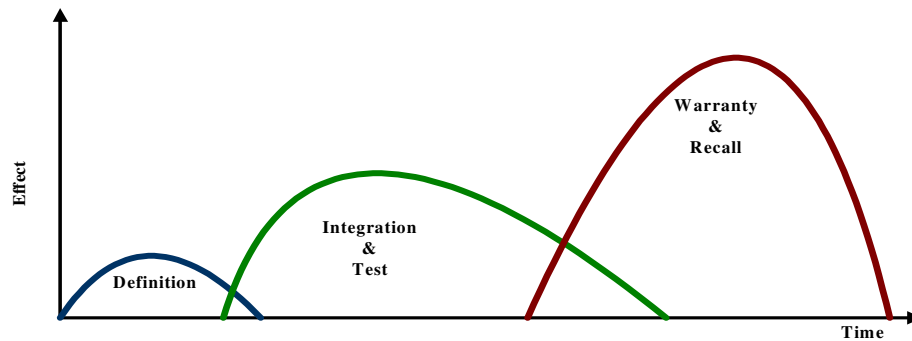


Figure 3.12: Traditional Network Design Process Effect

In advanced design process (WESTMANN, 2006), however, the effort, as shown in Figure 3.13, focuses more on the analysis and design, which results in less time and money wasted on integration and testing. Additionally the warranty and recall will be reduced as much as possible.

The advantages of the advanced design process are:

- The communication validation is more adequate.
- Real worst-case latencies are deterministic.
- Bus utilization is close to 100%.
- System reconfiguration is more flexible.
- Warranty issues are inherently decreased.
- Gateway timing is validated.

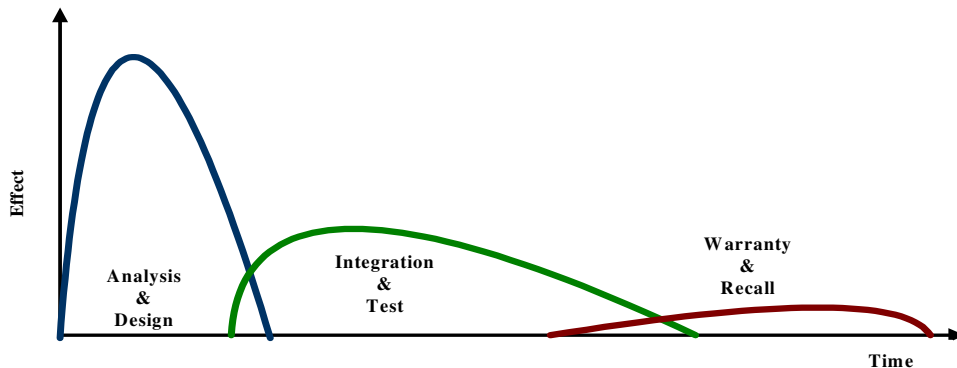


Figure 3.13: Advance Network Design Process Effect

V Model between Traditional and Advanced (HEURUNG)

The features of the Traditional V model, as numbered in Figure 3.14 include:

1. Incomplete network design and requirements analysis.
2. Engineers in this model assume the role of system integrator and node owner.

3. The configuration of the communication layer is based on incompletely validated information.
4. The ECU applications are dependent on network design parameters.
5. Issues identified during network verification require alteration, which involve the entire process flow and without proper validation can result in more network issues.

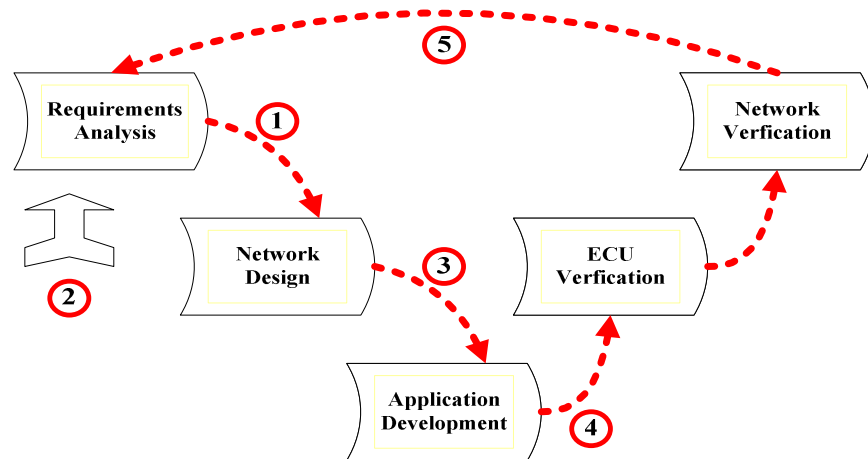


Figure 3.14: Traditional V Model

The features of the advanced V model, as numbered in Figure 3.15 include:

1. The requirements analysis and network design are more complete than traditional models.
2. Engineers in this model assume the roles of function owner, system integrator and node owner.
3. The configuration of communication layer is based on completely validated information.
4. The network communication design is more general.
5. ECU applications are independent of network design and topology.
6. The network verification occurs after the validation of implementation.
7. The network related issues are minimized as a result of upfront validation their changes does not involve entire process however, do not change the entire process.
8. The entire network design reconfiguration is very flexible.

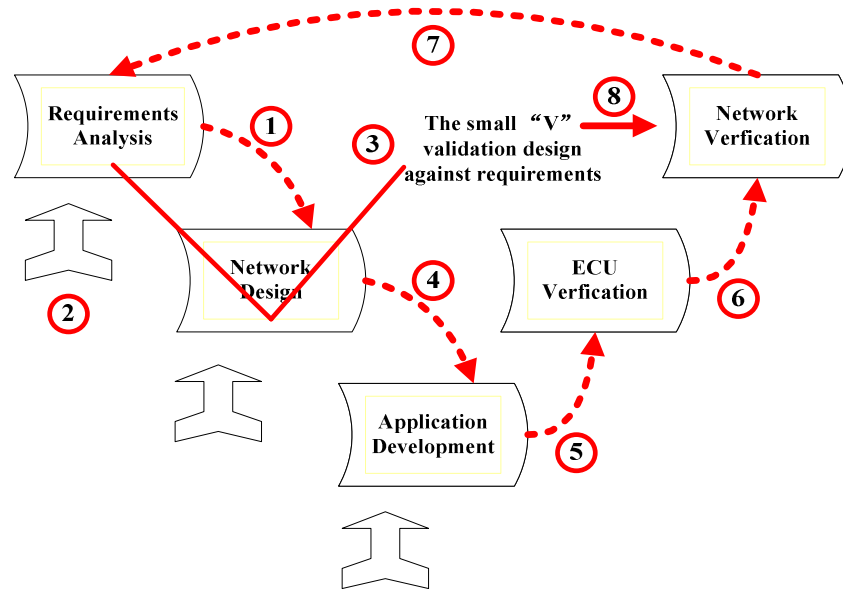


Figure 3.15: Advanced V Model

3.5 NETWORK MANAGEMENT

Networking an ECU into a vehicle should provide some certain features such as the accessibility of authorised entities, the tolerance of temporary failures, and the support of network diagnostics. As the number of ECUs supplied by various OEM in vehicle networks increases, a uniform network management mechanism is widely believed to guarantee the safe operation of safety-relevant, distributed systems (U., 1990). The main task (K.J, 1995) of the uniform network management in vehicles is to control the node operation modes, which include:

- Wake up the whole network system, mode switches from sleep to operation (Wakeup)
- Shut down the whole network system, mode switches from operation to sleep (Shut down)
- Mode switches between different application operations, such as initialization, operation, limp home.

3.5.1 OSEK/VDX

German motor companies such as Mercedes-Benz, BMW, Opel, French Renault motor company, and some OEMs such as Bosch and Siemens promote a special project, which is the co-establishment of an open architecture of automotive industry standard for distributed control equipments in vehicles.

This standard includes the Real-Time Operating System (OS), Network management (NM) and Communication API (COM). The first version of this standard was published in 1995 and called the OSEK (Open System and the corresponding interfaces for automotive electronics) / VDX (Vehicle distributed Executive System). In 2004, automotive electronic manufacturers and OEMs in Europe, established another version of OSEK/VDX 2.5.3, which is the latest version.

OSEK/VDX (LEMIEUX, 2001A) is a software interface for real-time systems in vehicles, which is defined as a communication and network management system. The OSEK operating system provides a set of services for node monitoring such as: Task

management, Synchronisation, Interrupt management, Alarms, Intra Processor Message Handling and Error Treatment.

3.5.2 OSEK/VDX NETWORK MANAGEMENT

The requirement for a communication network for each system and its associated sub-systems in vehicles is very rigorous (LEMIEUX, 2001A). To ensure safety and reliability of this communication network, it is essential to create a complete network management system. The major purpose of this operating system is to create a general platform combined with software modules from different manufacturers. This platform can also integrate all operation systems from a vehicle into different types of ECUs.

3.5.3 OSEK/VDX NETWORK MANAGEMENT COMPONENTS

The Network Management of the in-vehicle network environment was originally developed to be static, which means the overall nodes on the network are fixed and known by each other through the assigning of a unique identifier, although not every node needs to be available in an application. Figure 3.16 shows the in-vehicle network management environment (MORRISSEY, 2005).

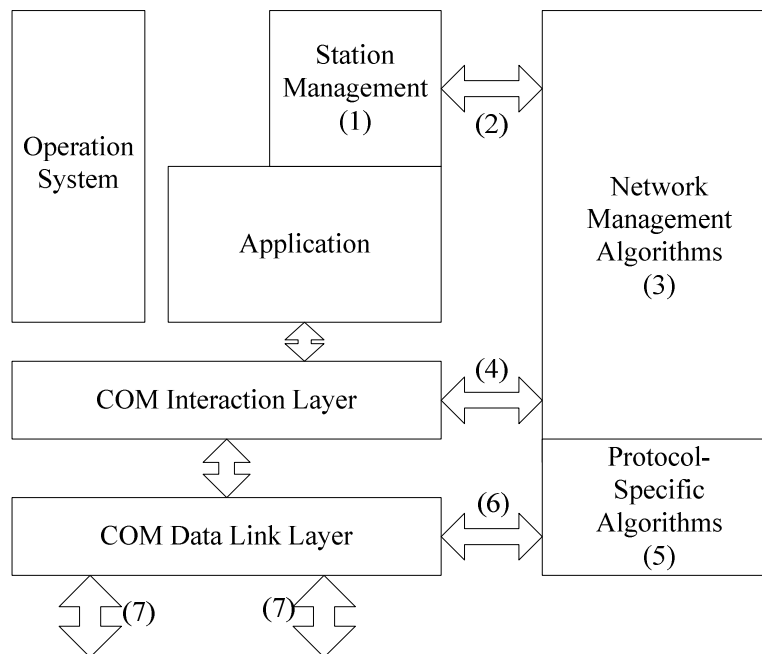


Figure 3.16: In-vehicle Network Management Environment

Network Management Methods

There are two network management methods, Direct and Indirect NM, defined in the NM specification (OSEK/VDX, 2004a). These two methods are classified by the way of monitoring their nodes.

In Direct NM, nodes on the network actively monitor each other. E.g. when a node checks another node, it will send a specific NM message to that node.

In Indirect NM, it involves the monitoring of periodic messages that are sent from all nodes.

In general, OSEK NM uses the specific NM communication to support direct node monitoring, which means it uses a node to communicate logically. From the OSEK stance, if a microprocessor with two communication models is connected to two different Communication Media, it represents two nodes, as shown in Figure 3.17.

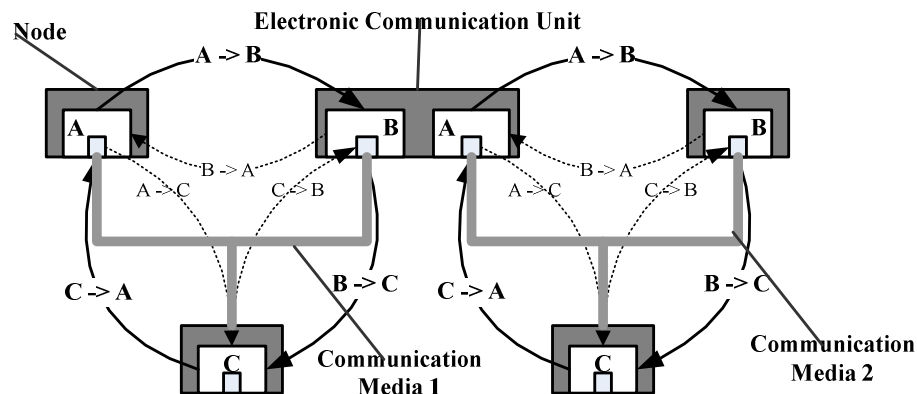


Figure 3.17: NM equipment with two CAN systems
(CAN-based Higher Layer Protocols and Profiles) (CiA, 2004)

3.6 CONCLUSION

There are no right and wrong methodologies for designing a network or system. The objective of the designer is to ensure that the solution meets the client requirements, is cost sensitive, provides revenue saving or generation and gives an avenue for future enhancements and growth.

For in-vehicle network designers, the key to network design is simple. Clearly define all requirements for the network. Fully understand the issues and problems concerning the network, both real and perceived. Use the interview and survey processes and listen very carefully to the users and operators. Therefore, the designers can reduce design time and increase quality. Some technologies, such as AUTOSAR discussed in Chapter 4, rely even more on reliable communication infrastructures that ensure data is delivered on time. AUTOSAR ultimately aims to reduce component cost; design and validation complexity will increase. Advanced tools and methodologies for in-vehicle network design have to be adopted to ensure “right first time” is possible.

3.7 REFERENCES

- AXELSSON, J. (1999)** Holistic Object-Oriented Modelling of Distributed Automotive Real-Time Control Applications. Volvo Technological Development Corp
- BJÖRN WESTMANN (2006)** Designing Networks vs. Testing Networks. ed., Mentor Graphics.
- BROWN, B. (1996-2000)** Networking Fundamentals Network Topology.
- CIA (2004)** CAN-based higher-layer protocols and profiles. ed., CAN in Automation.
- EASIS (2005)** Conceptual Hardware Architecture Specification, Version 1.0. EASIS - Electronic Architecture and System Engineering for Integrated Safety Systems
- HEURUNG, T.** In-Vehicle Network Design Methodology. Mentor Graphics
- INS (2002)** Basic Network Design Methodology. INS Whitepaper
- K.J, K. U. A. N. (1995)** Open System and Interfaces for Distributed Electronics in Cars (OSEK). SAE (Society of Automotive Engineers). Detroit, USA, SAE
- LEMIEUX, J. (2001)** Programming in the OSEK/VDX Environment, CMP Books; Pap/Cdr edition
- MORRISSEY, D. (2005)** OSEK/VDX Presentation. SAE (Society of Automotive Engineers). Automotive Control Group, Waterford Institute of Technology.
- NICOLAS NAVET, Y. S. (2005)** Trends in Automotive Communication Systems. The IEEE.
- OSEK/VDX (2004)** OSEK/VDX Network management Specification. Concept and Application Programming Interface. OSEK VDX Portal.
- S.BROWN, C. (2006)** In-vehicle Network Design Methodology. Mentor Graphics
- U., K. (1990)** Distributed Real-time Processing in Automotive Networks. SAE (Society of Automotive Engineers). Detroit, USA, SAE

NETWORK GATEWAY DESIGN

4.1 INTRODUCTION

There are lots of network protocols available in the automotive industry, each with their own advantages and disadvantages. It is impossible for a single protocol to satisfy all vehicle requirements. Therefore, gateways between different networks are necessary.

There are three main issues related to in-vehicle networks, which have to be considered when designing an in-vehicle network gateway. They are Safety/Non-safety, Time-triggered/Event-triggered and Secure/Non-secure (EASIS, 2005a).

Safety/Non-safety: Communication between different functional networks can cause faults on the safety-critical and dependability of the functional networks. It is necessary to handle connections crossing the borders of functional networks based on their dependability level. Faults in the non-safety network should not impact the safety network.

Time-triggered/Event-triggered: An in-vehicle network gateway has to deal with a vast amount of data with different types of protocols, which can be time-triggered (e.g. FlexRay), or event-triggered (e.g. CAN). In certain situations, if the network is not designed properly, the performance of the gateway can be compromised. Therefore, it is essential to pay attention to the interaction between the time-triggered and event-triggered buses at Lower Layers, by using dedicated hardware.

Secure/Non-secure: More and more Telematics' equipment are configured into vehicles to exchange information with external applications. These external communications require extremely secure communication to protect against any

intrusion, which maybe regarded as a threat to the entire in-vehicle network. For this reason, the gateway design should pay attention to both the hardware and software levels.

4.2 GATEWAY HARDWARE ARCHITECTURES

4.2.1 GENERAL GATEWAYS ANALYSIS

As mentioned in the introduction section, a gateway ECU has to address three issues, particularly on the inter-network communication between safety-critical and non safety-critical communications. Therefore, a basic structure, called Fail Silent Units (FSU) (TORIN, 2000), as shown in Figure 4.1, is necessary.

Fail Silent technique: “a node is designed so that it can only do correct sending of messages or no sending at all (be silent), which makes it possible to use only one bus (with redundancy for permanent failure resistance) for the inter-node communication.” (TORIN, 2000)

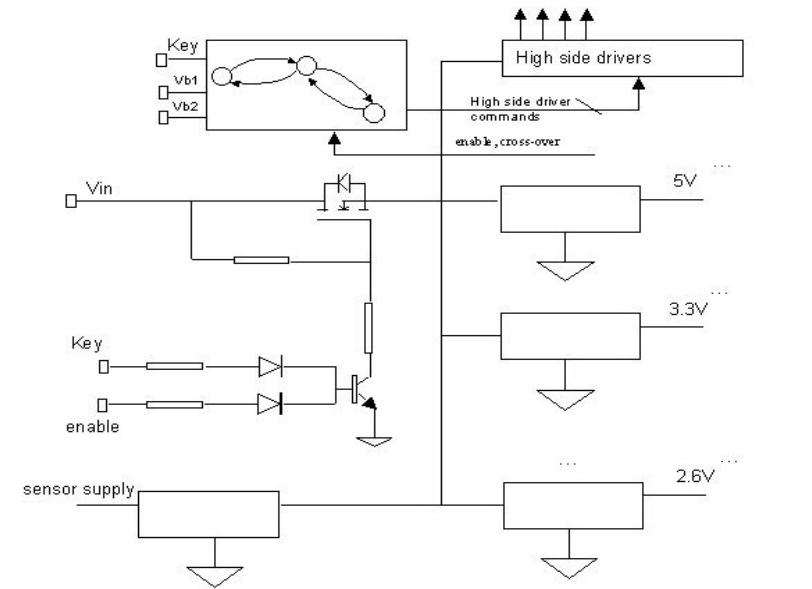


Figure 4.1: The Basic Structure of Fail Silent Units (FSU)

Figure 4.1 is an example of a basic block of a Fail Silent Unit (EASIS, 2005a). It includes all the regulators required to supply the FSU's internal circuitry. Moreover, it includes the high side drivers necessary to drive the power switches. A switch, once turned off, cuts power thus, permitting storage information in memory before complete switch off. Additionally, a hardware block implementing a state machine for controlling the switch in case of fault of one power supply is included. If one of the power supplies falls below a certain level, a fault on that line is detected; the state machine guarantees an immediate switch over.

The gateway ECU contains a bus guardian mechanism to control the correct node transmission performance in the safety network. The bus guardian should protect a faulty node from disturbing the whole network. It can enable and disable the bus-driver by using the communication schedule and allow each ECU access to the bus only during its own assigned time slots.

The bus guardian is located in the supervisor. The supervisor should have its own communication schedule and work independently from the communication controller. The supervisor will send an error signal to the main processor under two situations: the communication controller schedule is different from its own schedule, or the communication controller start cycle is different from the one calculated. Once an error signal is received, the main processor would ignore any further bus action.

4.2.2 *GATEWAY ARCHITECTURES*

There are three gateway hardware architectures (EASIS, 2005a): Single-processor gateway, Dual-processor with services separation and Dual-processor with domain separation.

Single-process Gateway: this is a gateway for safety critical/non-safety critical interfacing. It is a FSU concept with the requested communication interface(s) and the software. Figure 4.2 shows the signal-processor gateway block connecting two networks: safety-critical and non-safety critical. The single processor in this gateway processes all the software functions, such as communications, inputs/outputs and applications independently.

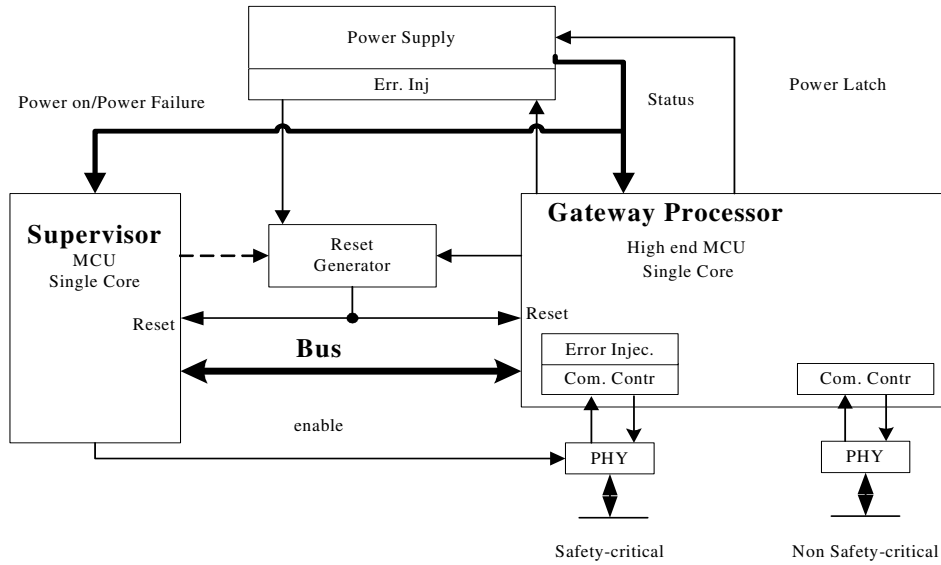


Figure 4.2: Single-process Gateway

Dual-processor with Services Separation: this is a gateway consisting of separated services through the use of two processors; one to execute the applications and I/O, and another to execute the Lower Layer communications. Figure 4.3 shows the dual processor with services separation gateway connecting three networks, one for safety critical, and the remaining two for non-safety critical. The communication processor implements routing services and other Low Layer communications. All Upper Layer services, such as NM, Upper Layer gateway services, remote access and firewall services are implemented on the application processor.

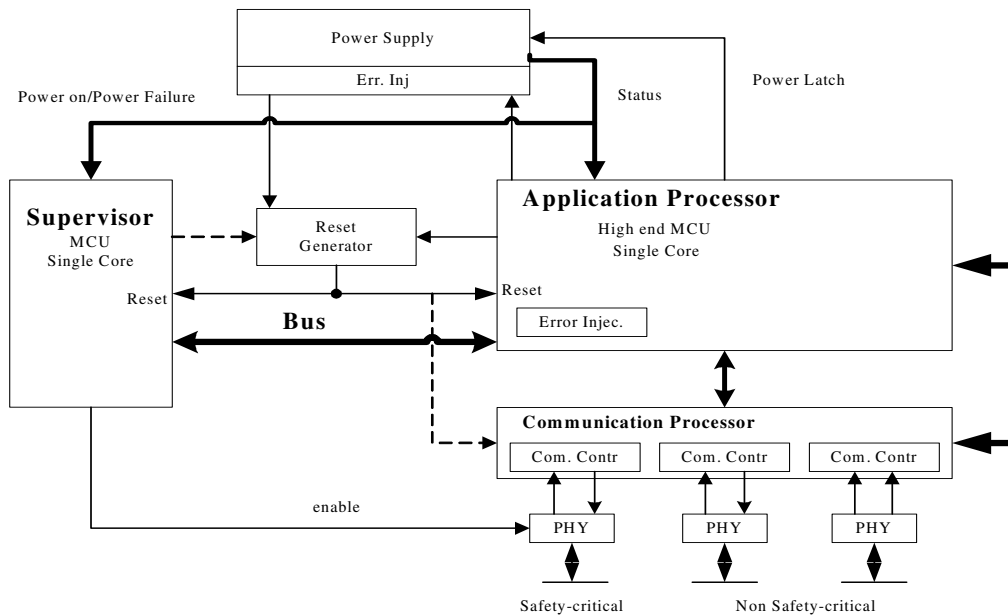


Figure 4.3: Dual-processor with Services Separation

Dual-processor with Domain Separation: is a gateway using a dual processor. In this gateway architecture, one of the processors is connected to safety critical/secure, while the other processor is connected to non-safety/non-secure domains. Application tasks are executed in one or other processors depending on its safety/security level.

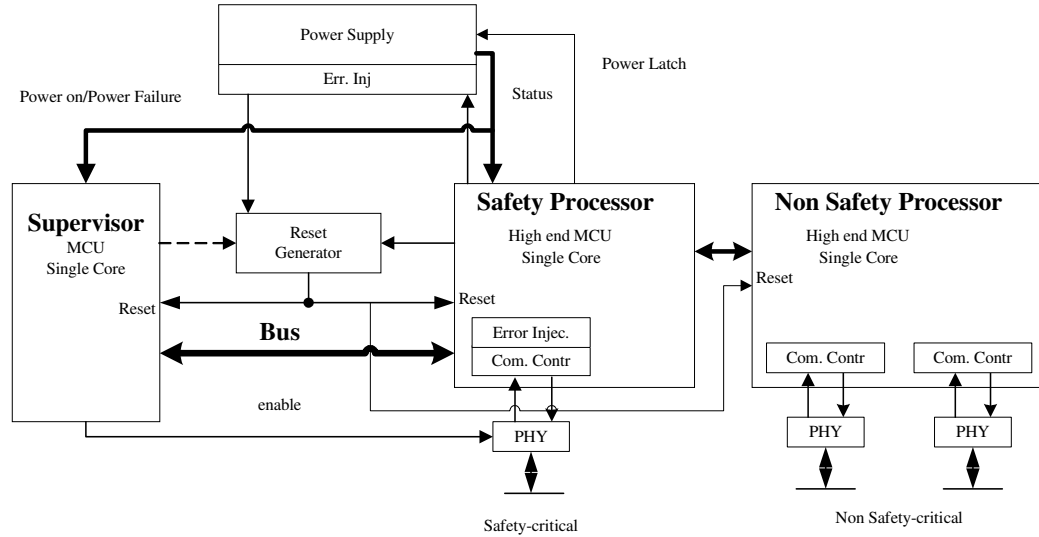


Figure 4.4: Dual-processor with Domain Separation

4.3 GATEWAY SOFTWARE MODULES

“In software gateway concepts, the focus is on the definition of a vehicle on-board electronic gateway software infrastructure that supports the requirements of integrated safety systems in its main aspects related to safety, security and reliability.” (EASIS, 2005a) Therefore, in this section, two general software modules are described.

OSEK COM (OSEK/VDX, 2004B)

The OSEK standard comprises an agreement on interfaces and protocols for in-vehicle communication called OSEK COM. OSEK COM provides a standardized API for the software communication that is independent of the particular communication media used in a way to ease porting of applications between different hardware.

The OSEK COM (LEMIEUX, 2001B) standard is composed of:

- An Interaction layer which provides communication services for the transfer of application messages.
- A Network layer which provides services for the unacknowledged and segmented transfer of application messages. The Network layer provides flow control mechanisms to enable interfacing of communication peers featuring different level of performance and capabilities.
- A Data link layer interface which provides services for the unacknowledged transfer of individual data packets over a network to the layers above.

AUTOSAR (AUTOSAR)

AUTOSAR was founded in response to the high cost of software. The cost is comprised of two factors: high development effort and error removal after start of production. In the automotive industry, “lack of reuse” means that similar functions are developed more than once and that functions are exercised with a frequency insufficient for confident quality assurance. AUTOSAR is based on existing standards, so that some existing implementations can be reused. Reusing software will reduce development costs, increase the use of modules, and reduce the amount of errors, all of which will result in fewer software-related recalls.

AUTOSAR defines the application environment (MORGAN, 2006), which defines a software component's interface to the rest of the ECU and the vehicle. The abstraction of the environment is called the Virtual Function Bus (VFB). The VFB acts as a communications matrix connecting I/O units and software components, as shown in Figure 4.5.

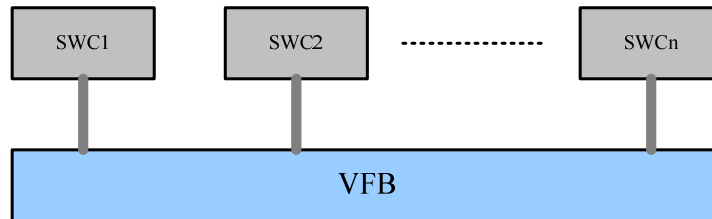


Figure 4.5: Software Components in the system communication via the VFB

In order to build a real system, all software components are connected to the AUTOSAR Runtime Environment (RTE) (MORGAN, 2006) to implement the VFB. The RTE, as shown in Figure 4.6, behaves like a telephone exchange, connecting software components, I/O units and other services. More details of AUTOSAR are discussed in a later chapter.

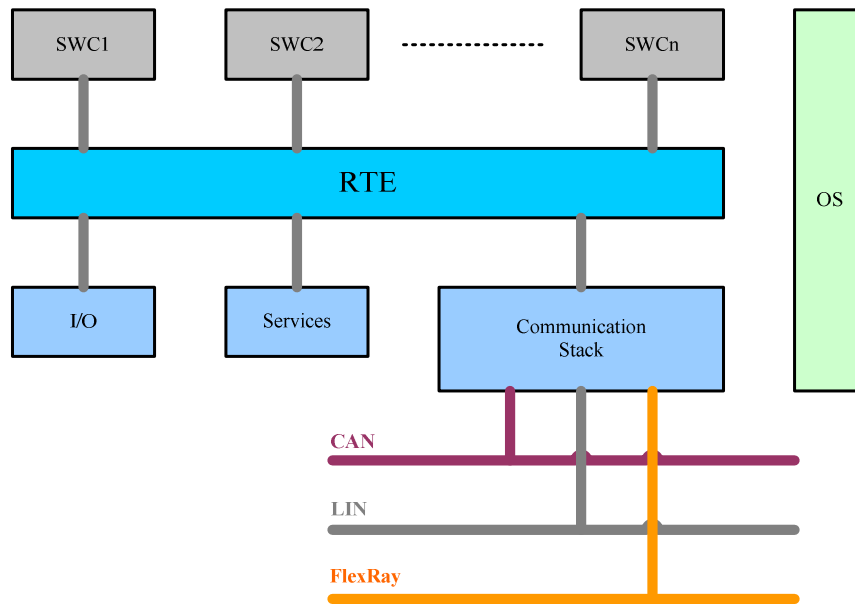


Figure 4.6: Software Architecture of an AUTOSAR ECU

4.4 OPTIMIZATION OF GATEWAY PERFORMANCE

4.4.1 USE OF ROUTERS FOR OPTIMIZING GATEWAY PERFORMANCE

The basic functionality of a gateway is “the ‘routing’ of interchanged data with appropriate performance to deal with the possibility of a huge interchange of information between the different networks (with different timings and payloads).” (EASIS, 2005a)

Optimising a gateways’ performance is to improve the routing function, so that it can efficiently:

- Reduce the delay between network communications.
- Increase network reliability and availability.
- Improve main or application ECU by off-loading communication functions.

There are two levels to improving the performance of a gateway:

A **signal gateway** is concerned with the signal level in a gateway. It usually:

- Communicates between functions with the exchange of signals.
- Forwards data with protocol independence.

A **message router** is concerned with the message level. It usually:

- Bridges different protocols.
- Reduces communication delay and increases network efficiency.

4.4.2 HARDWARE/SOFTWARE PARTITIONING AND ROUTING LEVELS

As mentioned earlier, a gateway has two levels, a signal level and a message level. The signal router at the signal level routes the individual signals or a group of signals between different COM modules, and the message router routes messages between different communication controllers (CAN, LIN, FlexRay, etc.). Furthermore, it is necessary to distinguish between “Signals”, “Message” and “Frame” during the operation of a gateway, as shown in Figure 4.7.

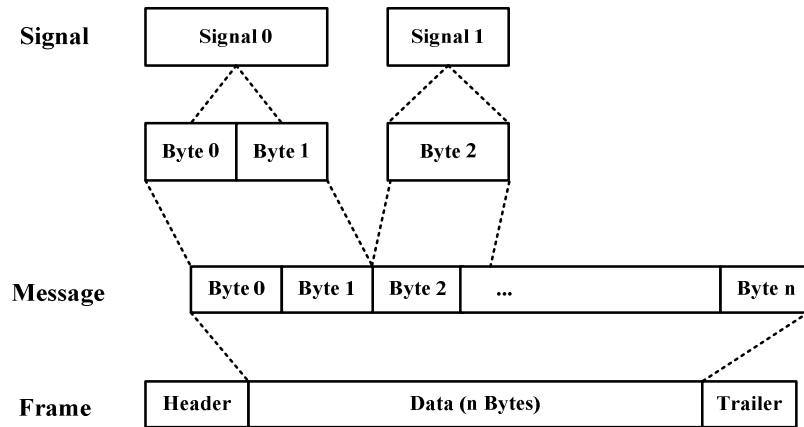


Figure 4.7: Message Definition

For each protocol stack with the length defined by protocol specification, signals from different API functions are mapped into messages. Data transmission on the bus is in the form of frames. Since each message regarded as a Protocol Data Unit (PDU) has to be configured with a header and a trailer to form a frame.

Recall how to the hardware/software partitioning is conducted in gateways; it is necessary to separate a single gateway ECU into a main processor and a communication process. This can also be related to **Computer Architecture** (STALLINGS, 2002).

Communication Processor: takes over the bus controller from the main processor, so that data exchange is implemented directly between memory and I/O interface without the main processor. This communication processor does the following work:

- Sends address and control signals to memory.
- Modifies data addresses.
- Counts the data length and
- Sends transmission end signal to main processor by using interrupts.

Main Processor: works as follows:

- Does not take charge of data exchange,
- Does not work as interrupts and
- Realises memory address modification and data transmission counting by hardware not software.

By using the communication processor and main processor separately, gateway performance and efficiency can be improved significantly. The dual processor gateway discussed in the hardware architecture is an example of a communication processor and main processor. Figure 4.8 shows the HW/SW partitioning of the dual processor gateway.

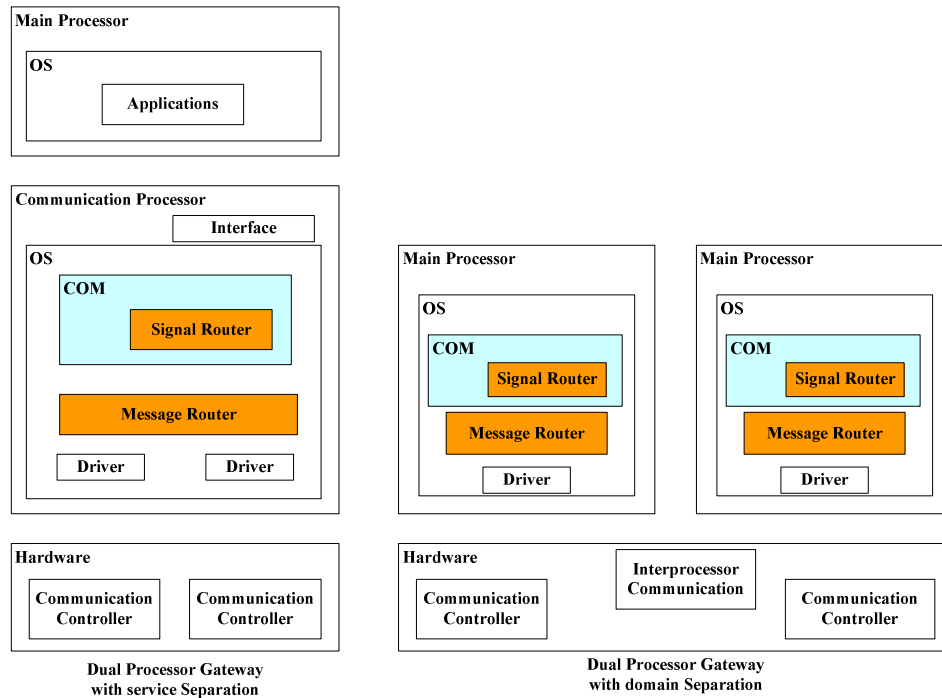


Figure 4.8: HW/SW partitioning of dual processor gateway

Figure 4.8 illustrates that the routers are more concentrated on software, but can also be supported by dedicated hardware, such as **DMA** (Direct Memory Access), with short connections or deterministic latency and/or very high reliability and availability. Figure 4.9 shows an example on this kind of gateway.

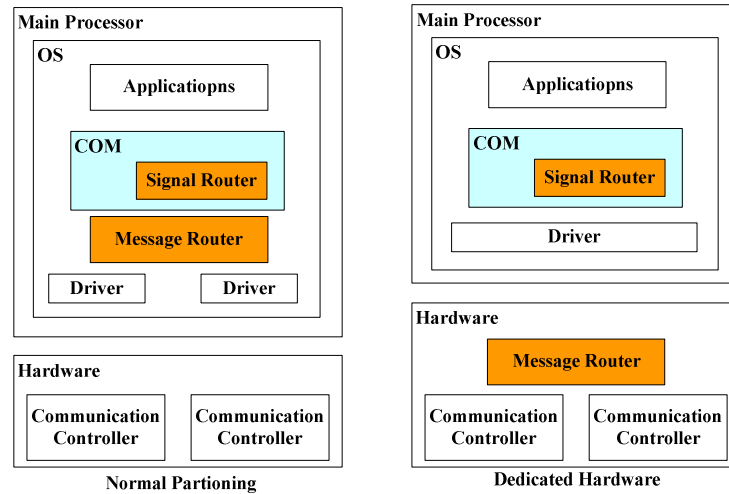


Figure 4.9: The Gateway Structure with a Dedicated Hardware

4.4.3 ROUTER STRUCTURE IN GATEWAYS

A basic gateway structure is shown in Figure 4.10. There are different protocol controllers configured in a vehicle network. It is essential to define a standard interface in this gateway, so that it can be reused by different protocol controllers.

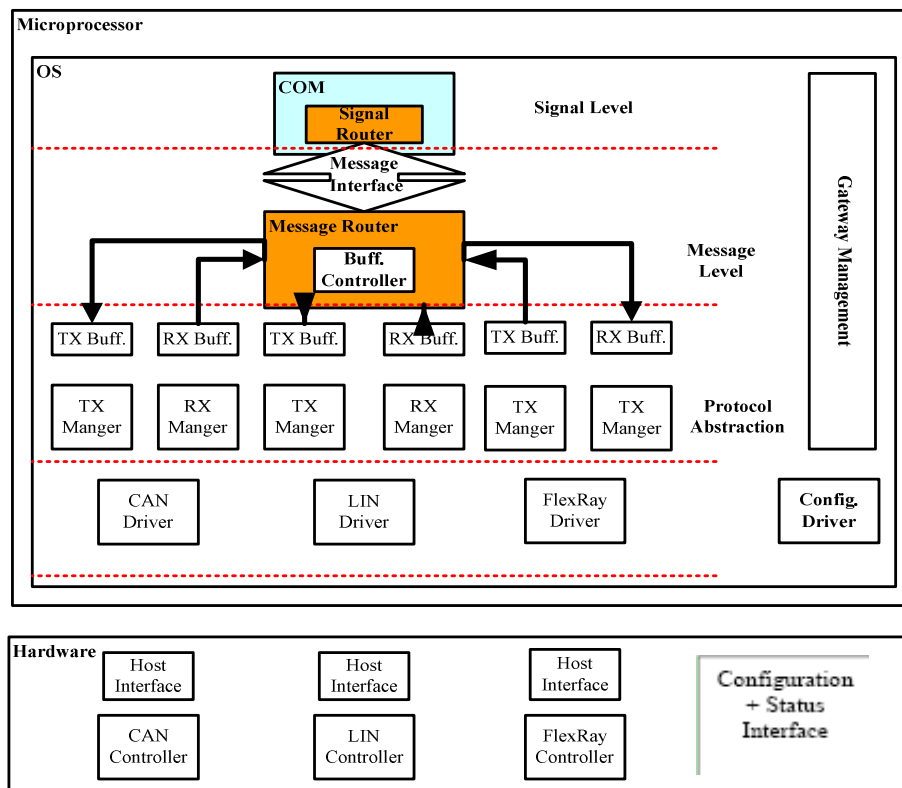


Figure 4.10: A Basic Gateway Structure

Router management in Software

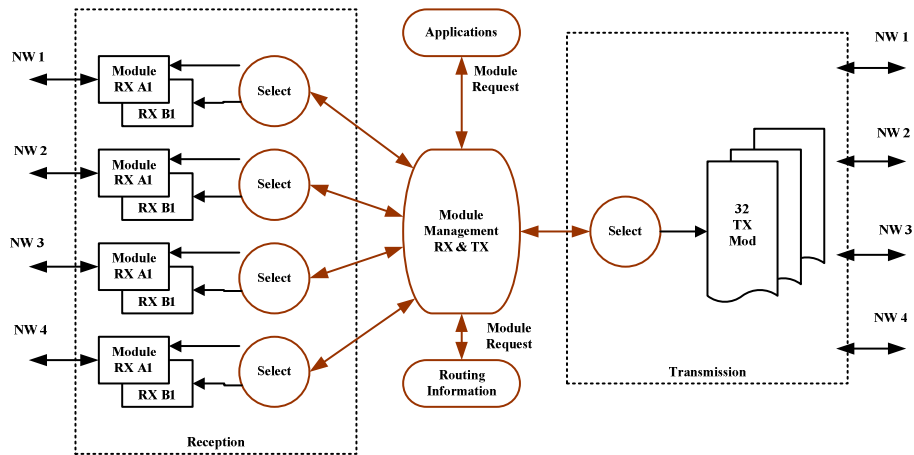


Figure 4.11: Router Management Structure

The software for a message router management, as shown in Figure 4.11, consists of five modules: Module Management, Routing Information, Reception, Transmit and Application.

Module Management: decides the whole router state, whether it is in receive or transmit.

If in receive state, the module manager has to look up the routing table to decide if an incoming message matches that shown. If incoming message matches, the module manager will inform the Select in Reception to accept that message, then route it to Application. If the message does not match, the module manager will inform the Select in Reception not to accept the message. Since it is a time-consuming process, the module manager avoids copying the entire incoming message data by handling the message address instead of the message content.

If in transmit state, the module manager has to work with the Select in Transmit to decide the message sequence according to their priority.

Routing Information: contains statically configured routing tables, which can be dynamically changed for certain parameters. Each table contains different information

on each message such as Frame ID, Network Number, Message Length, etc. The module manager will look up those tables information, when it is in receiving state.

Reception: contains a Select, which works as a filter to decide if it should let an incoming message pass or not. The Module Management controls those selected in Reception. Also a received message can be copied into multiple RX-modules in case it is routed to multiple networks (and/or the application).

Transmit: contains a select, which work with the Module Management to decide the messages priority to for transmission, according to their identifiers. All the transmitted messages prioritized will be assigned to appropriate buffers.

Application: requests the Module Management to send its data onto the network or execute its function once enabled by the Module Management.

4.4.4 DEDICATED HARDWARE FOR ROUTER

As discussed in the Hardware/Software Partitioning and Routing Levels section, a communication processor can be regarded as dedicated hardware for a message router, as shown in Figure 4.12. The following factors must be taken into consideration when using dedicated hardware:

- Appropriate buffer depth with efficient bus arbitration mechanism can improve the reliability and the speed for the router. Discussed in gateway design section.
- Task scheduling and prioritization of the routing functions can be well configured by the operation system such as OSEK.
- The same as in software, the dedicated hardware should also apply good routing functions such as filter mechanism and routing tables.

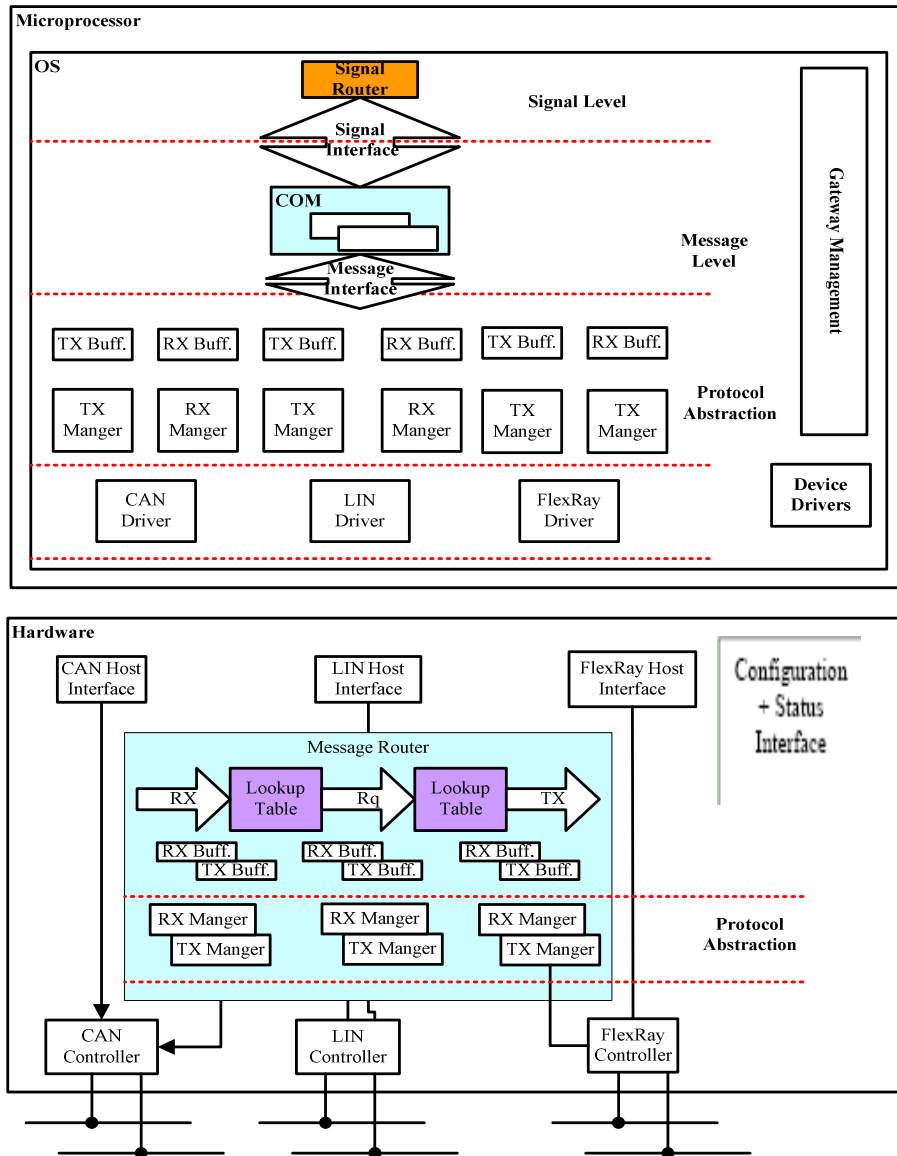


Figure 4.12: Router structure with hardware support

The message router in a dedicated hardware can work autonomously and independently from that hardware. In Figure 4.13, a hardware message router has almost the same functions as a software router.

A well-selected dedicated hardware can work as a message router more effectively than software, because the delay in code execution does not occur in hardware. Figure 4.13 shows the basic structure of a router with message router hardware support for a single processor gateway.

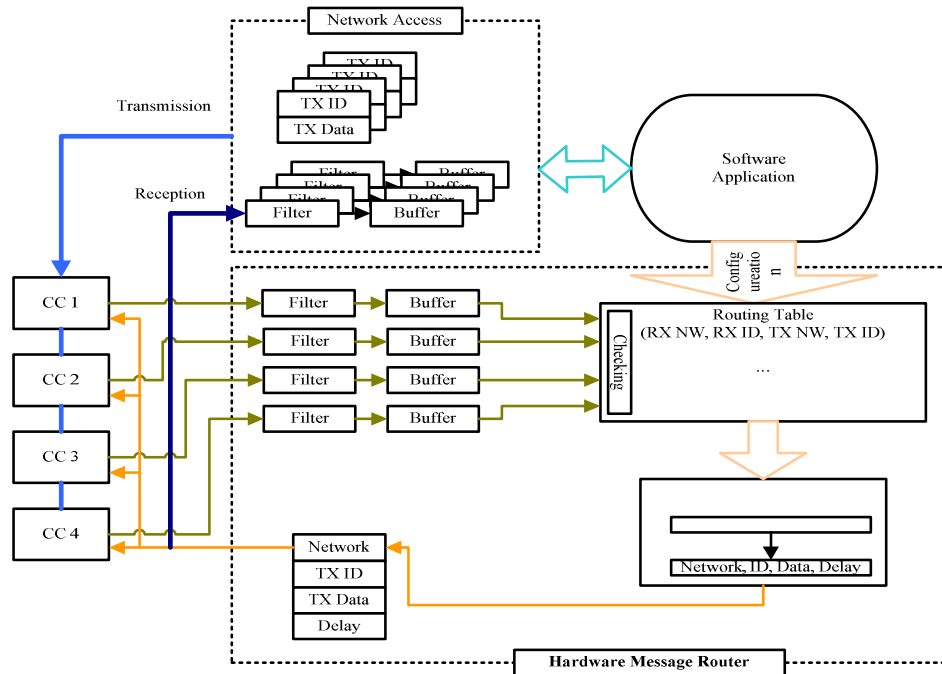


Figure 4.13: The Structure of a Hardware Router

4.4.5 PROTOCOL ANALYSIS

As discussed previously, message routers at a low level of the protocol stack must convert different protocol messages and bridge them between either sides of a gateway.

It is necessary to define an interface at the bottom of a communication protocol stack that can carry out conversions between different communication protocols.

Table 4.1 shows the difference between some protocols:

	Speed (Kb/s)	Payload Size (Bytes)	Access Protocol
LIN	20	8	CSMA-CA, Time-triggered
CAN B	125	8	CSMA-CD, Event-triggered
CAN A	1000	8	
FlexRay	10000	254	TDMA and FTDMA, Time-triggered

Table 4.1: In-vehicle Network Protocol differences

During message transmission and reception, if message routing is required between networks with the same protocols such as CAN \Leftrightarrow CAN and FlexRay \Leftrightarrow FlexRay, the message can be implemented without protocol abstraction.

But if message routing is required between networks with different protocols such as CAN \leftrightarrow FlexRay, the message must be implemented with protocol abstraction. Examples of different protocol network communication are discussed next.

FlexRay \leftrightarrow FlexRay

Using a gateway to connect two or more FlexRay networks can be achieved by applying compatible configurations on the networks.

CAN \leftrightarrow FlexRay (Dynamic Segment)

This connection case requires low deterministic timing, latency, reliability and robustness, so that the dynamic segment of FlexRay is the better option. The protocol allows for configuration and dynamic adaptation of the frame towards the needs of a common interface. However, if larger FlexRay frames are allowed by the configuration, some additional facilities will be necessary.

CAN \leftrightarrow FlexRay (Static Segment)

If the communication between a CAN bus and a FlexRay bus requires deterministic timing, latency, reliability or robustness, the static segment of FlexRay should be chosen. For the purpose of high requirements, this protocol requires more measures in the abstraction layer.

Figure 4.14 shows how this communication works.

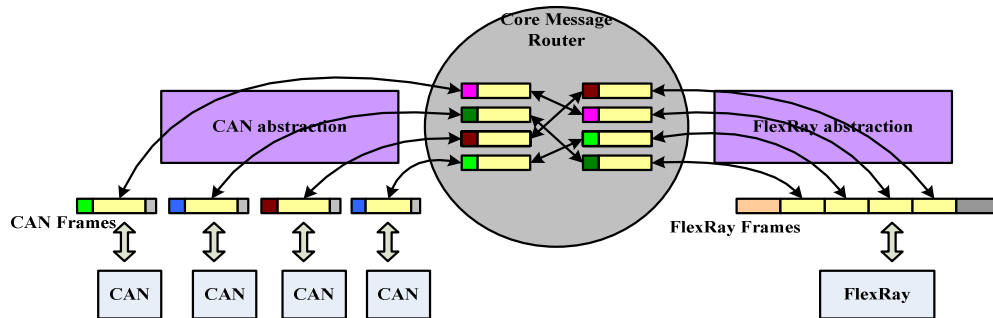


Figure 4.14: CAN and FlexRay Communication

4.5 CONCLUSION

Ideally, dedicated hardware, such as DMA can be used for data transmission between a gateway interface and memory or other interfaces. Once a processor is configured, the DMA controller implements the data transmission, and the embedded processor implements the application data.

Engineers have several design options, but the most important for consideration is whether the DMA controller is needed. It depends on the structure of the embedded processor. For implementing data transmission application, the processor might have enough idle cycle. But to minimize the data transmission cycle, it is necessary to consider the transfer capacity and DMA transmission to get every cycle for the processor.

Thus, the in-vehicle network gateway structure needs to consider several factors. It has to meet the requirements of bandwidth and latency requirements for its interface. The design of the gateway must be customizable, inexpensive and robust in order to maintain competitiveness.

4.6 REFERENCES

AUTOSAR, AUTomotive Open System ARchitecture.

EASIS (2005) Conceptual Hardware Architecture Specification. Version 1.0 ed.,
Electronic Architecture and System Engineering for Integrated Safety Systems.

LEMIEUX, J. (2001) Programming in the OSEK/VDX Environment CMP Books.

MORGAN, G. (2006) AUTOSAR -- Content, Methods, and Applications.

LiveDevices, ETAS Group.

OSEK/VDX (2004) OSEK/VDX Network management Specification. Concept and
Application Programming Interface. OSEK VDX Portal.

STALLINGS, W. (2002) Computer Organization and Architecture: design for
performance, Prentice Hall.

TORIN, Ö. A. A. J. (2000) Fail-Silent Computer Node. Department of Computer
Engineering, Chalmers.

5.1 INTRODUCTION

Simulation is the study of developing a dynamic model for a system, and it is the process of applying the research results, based on experiments and investigations, to a real-world system, such as a production system manufacturing automobiles. Here, the system is the objective of research, the model is the description of the system, the simulation is the tool and method of system investigation, and these three are closely related.

System

A system is a group of entities that are combined in some regular interaction or interdependence toward the accomplishment of some purpose. For example, an in-vehicle network system, different protocols, software and hardware are integrated together to produce a complete network system.

Systems, according to their performance characteristics can be categorized as discrete or continuous. “Few systems in practice are wholly discrete or continuous, but since one type of change predominates for most systems, it is usually possible to classify a system as being either discrete or continuous.” (LAW, 2000)

A system is discrete, if its state variable(s) change only at a discrete set of points in time, and are driven by stochastic events. (BANKS, 2000B) The vehicle network is an example of a discrete system, since the state variable, that is, the number of messages in the network, changes only when a message arrives or when the service providing that a message is complete. Figure 5.1 shows how the number of messages changes only at discrete points in time.

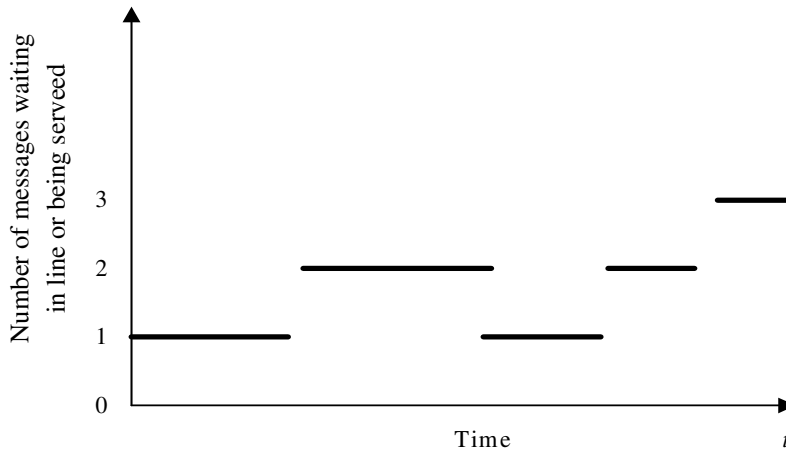


Figure 5.1: The Example of a Discrete System

A system is continuous, if its state variable(s) change continuously over time. A water dam is an example of a continuous system. (BANKS, 2000B) Water is drawn from the dam for flood control and to make electricity. Figure 5.2 shows how the state variable, that is, the head of water behind the dam, changes for this continuous system.

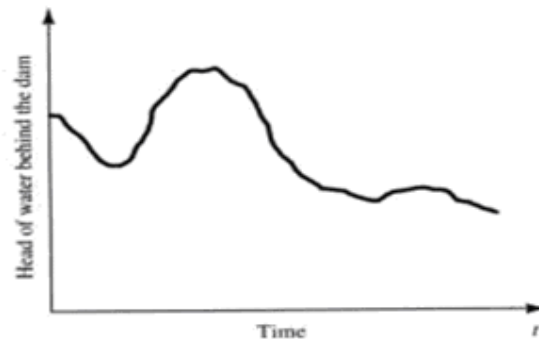


Figure 5.2: The Example of a Continuous System

Model

A model is the description of a real-world system. A model is the simplification and substitution of the system and it must include the major features of this system. In generally, models can be classified as being mathematical or physical.

Physical models magnify or minimise the real-world system to certain proportions. Mathematical models use symbolic notation and mathematical equations to represent a system. A simulation is a particular type of mathematical model. Simulation models have different types.

A simulation is static, if the state variable(s) in its model have no timing factors, otherwise a simulation is dynamic. Discrete and continuous systems introduced earlier are also types of simulation models.

System Simulation (ARSHAM, 1995)

To analyse, investigate, integrate and design a system, it is essential to conduct experiments on it. Sometimes it is possible to carry out experiments directly with the system itself. It is also possible to do experiments by creating a model based on the system to be investigated.

Here are some problems, when conducting experiments on a real-world system:

- They can destroy the system operation;
- They can delay the system design cycle;
- They can hinder recovery to the original system;
- They may not provide accurate judgments and estimations under different conditions each time.

The simulation process creates a system model and a simulation model, whilst conducting experiments based on these models.

5.2 SIMULATION METHODOLOGIES

Models, according to their types, can be simulated by two methods: Physical Simulation and Mathematical Simulation. (GOULD, 2006)

Physical Simulation

Physical simulation is the testing and studying process that creates a physical model for the system to be simulated in terms of its physical features. The advantages of a physical model are intuitional and visual. However, creating a physical model requires high investment cost and time. Conversely, with a physical model, it is difficult to modify the system structure, so that the research will be restricted.

Mathematical Simulation

Mathematical simulation is the testing and studying process that creates a mathematical model for the system to be simulated in terms of its relations. Mathematical models are economical, convenient, time saving and flexible. The steps of simulating a mathematical model are (RU, 1995):

1. Converting a mathematical model to a simulation model, which can then be studied on a computer;
2. Solving the simulation model on a computer;
3. Carrying out analysis and study for this system.

Mathematical simulations can not work without computers, so a mathematical simulation is called a computer simulation. Computer simulations are used to create a mathematical model by using computers for analysis and study, and then applying the analysis and results of the study to the real system. Thus, a computer simulation includes three essentials: systems, models and computers, and these three essentials are associated by three basic activities: model establishment, simulation establishment and simulation experiments.

5.3 SIMULATION TECHNOLOGY

Queuing Theory, that is, the so called Stochastic Process, is widely used in different fields, such as communication networks, computer systems and machine plants. Queuing Theory is considered as applicable for data transmission analysis in vehicle networks and gateways.

When the process of queuing happens, the part that wants to get service is called a customer. On the other hand, the part, which provides service, is called a server. As customers, they want to get service in time whilst waiting as little time as possible. A busy state on the server is regarded as better than an idle state, as being in a busy state increases the usage factor.

The preceding sections will demonstrate how queuing theory is applicable to the data transmission in the vehicle network and gateways.

5.3.1 THE BASIC QUEUING THEORY

Figure 5.3 is a general model of a queuing process. It illustrates that before a message from source goes to a certain server, it will wait in the queue to be served according to the queuing discipline. The server will carry out the service following the service discipline, and a message will leave after being served. The queuing structure in Figure 5.3 describes the number of the queues and their ranking methods. The queuing and service disciplines describe the workings of the queuing structure. The service window provides service according to the queuing and service disciplines.

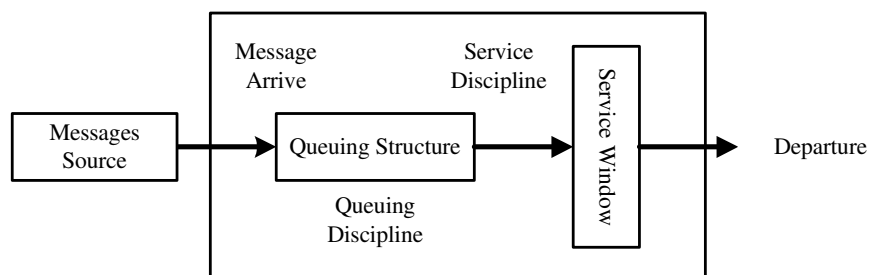


Figure 5.3: Basic Model of Queuing Process

Essentials of Queuing Theory (BANKS, 2000B)

Generally, a queuing system should be provided with three essentials, which are customers, queuing discipline and servers.

1. Customer

The term “customer” can refer to people, machines, trucks, mechanics, patients, pallets, airplanes, e-mails, cases, orders, or dirty clothes – anything that arrives at a facility and requires service. When designing an in-vehicle gateway, all the messages coming from different transceivers are said to be customers.

The customer sources and the states of queuing systems are various. The sources of customer could be finite or infinite. Customers can arrive continuously or discretely, they can come one by one or batch by batch. The interval time between customers’ arrival could be determinate or stochastic. Each customer arriving could be independent or interrelate with others.

The process of customers queuing is also called input process. If, however, the intervals between arriving customers, as well as their associated parameters are in no way affected by timing, we say this process is stationary, otherwise it is non-stationary. In general, some certain mathematical models can resolve a stationary input process, but for a non-stationary input process, only simulation models can resolve it.

2. Queuing Discipline

Losing System and Waiting System

When customers arrive and all the servers are busy, customers can leave or wait. ‘Customers leaving’ is a losing system, and ‘customer waiting’ is a Waiting System.

Mixed System

Queuing systems in in-vehicle network gateways are between a Losing System and a Waiting System. When a message arrives, if all servers are busy it will queue; if all servers are busy and every queue is full, this message will leave.

Queuing Discipline

- **FIFO:** (first in, first out): a customer that finds the service centre busy goes to the end of the queue.
- **LIFO:** (Last in, First out): a customer that finds the service centre busy proceeds immediately to the head of the queue. They will then be served next, provided that no further customers arrive.
- **Random Service:** the customers in the queue are served in random order.
- **Round Robin:** every customer gets a time slice. If their service is not completed, they will re-enter the queue.
- **Priority Disciplines:** every customer has a (static or dynamic) priority; the server always selects the customers with the highest priority. This scheme can use pre-emption or not.

3. Server

The term “server” might refer to receptionists, repairpersons, mechanics, tool-crib clerks, medical personnel, automatic storage and retrieval machines (e.g., cranes), runways at an airport, automatic packers, order pickers, CPUs in a computer, or washing machines – that is, any resource (person, machine, etc.) which provides the requested service. In vehicle networks, gateways are servers that decide how to process messages.

Servers can be single or multiple. Multi-servers can be serial or parallel. The services the server provides can be by single or by batch; the service time can be deterministic, or stochastic, the distribution of service time can be stationary, or non-stationary.

5.3.2 *MARKOV PROCESS AND ITS MAJOR MODELS*

5.3.2.1 *MARKOV PROCESS*

The queuing process (JACOBSEN, 2006) is a stochastic process, while a queuing problem is a stochastic problem. Customers can come into server system at different interval times. The service time each customer requires is non-deterministic. Thus, some times the server can be busy, while sometimes it can be idle. It is quite complex to solve a stochastic problem than to solve a deterministic problem, because building

mathematical models is not easy. In some circumstances however, it is possible to build a mathematical model with simple structure and certain instructions. The Markov process is based on this kind of mathematical model.

Markov was a Russian mathematician, who after conducting lots of experiments discovered that under some circumstances, the probabilities in the system states' transfer process; that is, the probability of n^{th} result does not depend on the previous results, but usually on the $(n-1)^{\text{th}}$ result.

It is essential to know what the general stochastic process is before building a Markov process. In nature, the process of system transferring can be classified into deterministic and stochastic. If the process of system transferring has a certain format, we say this process is deterministic. A deterministic system can be predicted. If the process of system transferring cannot be predicted, we say it is a stochastic process.

The basic concept of the Markov process is that of the system states and system transferring. Essentially, a variable of the system state transfers from one certain value to another certain value, thus, we say this system carries out a 'transferring'. For example, all the CAN nodes in an in-vehicle network functioning correctly represent one state, while state transferring happens when one or more CAN nodes are disconnected.

So the Markov Process's definition is:

The future probabilities of a random process are determined by its most recent values.

A stochastic process $x(t)$ is called Markov if for every n and $t_1 < t_2 \dots < t_n$, we have:

$$P(x(t_n) \leq x_n | x(t_{n-1}), \dots, x(t_1)) = P(x(t_n) \leq x_n | x(t_{n-1})). \quad (\text{WEISSTEIN})$$

This is equivalent to:

$$P(x(t_n) \leq x_n | x(t) \text{ for all } t \leq t_{n-1}) = P(x(t_n) \leq x_n | x(t_{n-1})) \quad (\text{WEISSTEIN})$$

5.3.2.2 DISCRETE AND CONTINUOUS MARKOV PROCESS

A discrete and continuous Markov Process provides big effects in Queuing Theory. A *discrete system* is one in which the state variable(s) change only at discrete points in time. The Chinese takeaway is an example of a discrete system, since the state variable, the number of customers in the takeaway, changes only when a customer arrives or when they have received their takeaway, that is, the service is complete. Figure 5.1 shows how the number of customers changes only at discrete points in time.

A continuous system is one in which the state variable(s) change continuously over time. For example, one mechanical system has two machine tools, where a machine tool could fail at any time. Once failure happens, it must be repaired at once. The time taken to repair will be non-deterministic, before failure happens, or it is stochastic.

The possible states of this mechanical system could be:

- S_0 : Both machine tools work properly.
- S_1 : Machine tool 1 works properly, but machine tool 2 does not.
- S_3 : Machine tool 2 works properly, but machine tool 1 does not.
- S_4 : Both machine tools do not work properly.

In fact, the transfers of system states are instantaneous. Namely, Faults and repairing are both stochastic.

5.3.2.3 EXAMPLES OF QUEUING MODELS

D.G.Kendall's classified a queuing system model (KENDALL, 1953). The format of D.G.Kendall's classification is as follow:

$$X/Y/c/N/K$$

X represents the inter-arrival-time distribution.

Y represents the service-time distribution.

c represents the number of parallel servers.

N represents the system capacity.

K represents the size of the calling population.

For example, $M/M/1/\infty/\infty$ indicates a single-server system that has unlimited queuing capacity and an infinite population of potential arrivals. The inter-arrival times and service times are exponentially distributed.

The data transmission in the vehicle networks can be modeled as a single server queuing system with a messages arrival process and an exponential distribution for service time. The gateway is the server and all waiting messages form queues. There are some common queuing models with Markov Process (JACOBSEN, 2006) give a general idea on how the single server works and how messages queue to make the analysis simple.

1. Single-Server Queues with Losing (M|M|0|0)

There is only one server in this system, so this system has only two states:

S_0 : Server is idle.

S_1 : Server is busy.

The messages arrival rate is λ , and the server's service capability is μ , as shown in Figure 5.4.

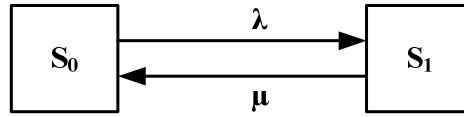


Figure 5.4: M|M|0|0 States

2. Single-Servers Queuing with Waiting (M|M|1)

This queuing model is very common in vehicle networks. The messages arrival rate is λ , and the server's service capability is μ , as shown in Figure 5.5. There is only one server in the system. When messages arrive, if the server is busy, they join the queue to be served.

S_0 : There is no message in the system, and the server is idle.

S_1 : There is only one message in the system, the server is busy, and the queue is empty.

S_2 : There are two messages in the system, the server is busy, and one message is in the queue.

...

S_k : There are k messages in the system, the server is busy, and there are $(K-1)$ messages in the queue.

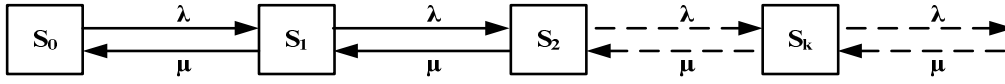


Figure 5.5: M|M|1 States

3. Single-Server Queuing with Losing and Waiting (M|M|1|m)

There is only one server in the system, the messages arrival rate is λ , and the server's service capability is μ , as shown in Figure 5.6. When a message arrives and the server is busy, that message must wait to be served. As the queue size in the system is m , if the queue is full, messages have to leave.

S_0 : The server is idle.

S_1 : The server is busy, and there is no message in the queue.

S_2 : The server is busy, and there is one message in the queue.

...

S_k : The server is busy, and there are $k-1$ messages in the queue.

...

S_{m+1} : The server is busy, and there are m messages in the queue.

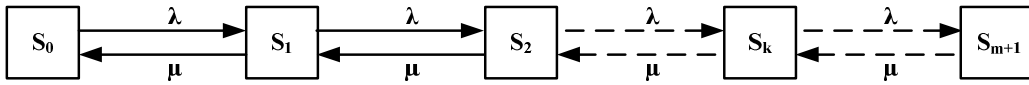


Figure 5.6: M|M|1|m States

5.4 SIMULATION TOOL – MATLAB/SIMULINK AND SIMEVENTS

5.4.1 BACKGROUND

The name Matlab derived from a combination of the first three letters of both MATrix and LABoratory. In the late of 1970s, Cleve Moler, the head of the Computer Science Department of University of New Mexico designed an easy understood interface using LINPACK (DONGARRA, 1980s) and EISPACK (WILKINSON, 1972-1973) software packages.

LINPACK is a collection of FORTRAN subroutines for solving linear equations and EISPACK contains subroutines for solving Eigen value problems.

After widespread used in universities and after a suggestion by John Little, Little, Moler, and Steve Bangert started working together. In 1984, MathWorks Inc. was founded and officially introduced to the market as Matlab (MATHWORKS, 1984).

As a software package of Matlab, Simulink is an integrated interactive environment for modelling, simulating and analyzing dynamic systems.

5.4.2 THE MATLAB SYSTEM

“Matlab is a high-performance language for technical computing. It integrates computation, visualization and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Modelling, simulation and prototyping,
- Math and computation,
- Data analysis, exploration and visualization,
- Algorithm development,
- Application development, including graphical user interface building,
- Scientific and engineering graphics and
- Data acquisition. (MATHWORKS, 1997B)

Matlab is an interactive system with dynamic basic data element array sizes. This allows for an easy interactive calculation of technical computing problems, especially those with vector and matrix formulations. To develop the same calculation and write the program in a non-interactive scalar language would take a much longer period of time and may prove very difficult.

The Matlab system consists of five main parts: Development Environment, The Matlab Mathematical Function Library, The Matlab Language, Graphics and Visualization, and External Interfaces/API.

The Development Environment, like the majority of development environments includes:

- The Matlab desktop and command window,
- A command history,
- An editor and debugger and
- Browsers for viewing help, the workspace, files and the search path.

The Matlab Mathematical Function Library collects computational algorithms from simple functions such as sum, sine, cosine and complex arithmetic to more difficult functions such as matrix inverse, matrix eigenvalue and Fourier transforms. (MATHWORKS, 1997B)

The Matlab Language is a high-level matrix/array language with control flow statements, functions, data structures, input/output and object-oriented programming features. It supports both small and complex application programs.

When using the Matlab Language, there are fifteen fundamental data types. Each of these data types is formed by a matrix or on array. The matrix or array is a minimum of 0-by-0 in size and can grow to an n-dimensional array of any size. (MATHWORKS, 1997B)

The following Table 5.1 describes these data types in detail.

Data Type	Example	Description
logical	<code>magic(4) > 10</code>	Logical array. Must contain only logical 1 (true) and logical 0 (false) elements. (Any nonzero values converted to logical become logical 1.) Logical matrices (2-D only) may be sparse.
char	<code>'Hello'</code>	Character array (each character is 16 bits long). This array is also referred to as a string.
int8, uint8, int16, uint16, int32, uint32, int64, uint64	<code>uint8(magic(3))</code>	Signed and unsigned integer arrays that are 8, 16, 32, and 64 bits in length. Enables you to manipulate integer quantities in a memory efficient manner. These data types cannot be used in mathematical operations.
single	<code>3*10^38</code>	Single-precision numeric array. Single precision requires less storage than double precision, but has less precision and a smaller range. This data type cannot be used in mathematical operations.
double	<code>3*10^300</code> <code>5+6i</code>	Double-precision numeric array. This is the most common MATLAB variable type. Double matrices (2-D only) may be sparse.
cell	<code>{17 'hello'</code> <code>eye(2)}</code>	Cell array. Elements of cell arrays contain other arrays. Cell arrays collect related data and information of a dissimilar size together.
structure	<code>a.day = 12;</code> <code>a.color = 'Red';</code> <code>a.mat =</code> <code>magic(3);</code>	Structure array. Structure arrays have field names. The fields contain other arrays. Like cell arrays, structures collect related data and information together.
function handle	<code>@humps</code>	Handle to a MATLAB function. A function handle can be passed in an argument list and evaluated using <code>feval</code> .
user class	<code>inline('sin(x)')</code>	MATLAB class. This user-defined class is created using MATLAB functions.
java class	<code>java.awt.Frame</code>	Java class. You can use classes already defined in the Java API or by a third party, or create your own classes in the Java language.

Table 5.1: Data Types in Details

Graphics are the extensive facilities of MATLAB which use graphs to display vectors and matrices and for annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation and presentation graphics. It also includes low-level functions that allow users to fully customize the appearance of graphics, as well as to build complete graphical user interfaces for your MATLAB applications. (MATHWORKS, 1997B)

External Interfaces/API allows users to interact with MATLAB by writing C and FORTRAN programs. It includes facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

5.4.3 SIMULINK AND STATEFLOW

Simulink is a block-based design environment based on Matlab for modelling, simulating, and analyzing dynamic systems. It has a comprehensive range of blocks to model any system represented by math, including linear and nonlinear systems, time driven and event driven system. Simulink provides plenty of functional models and different domain models to build a whole dynamic system without writing any code.

Stateflow is an interactive design tool based on Finite-state Machines (MathWorks, 1997b) to model and simulate some complex event driven systems. Stateflow is integrated with Simulink and Matlab. It is possible to link some complex control logic created by Stateflow to Simulink models.

5.4.4 SIMEVENTS

SimEvents (MATHWORKS, 2006) is an extensive tool of Simulink for modelling and simulating discrete-event systems using queues and servers. SimEvents allows users to build a discrete-event simulation model in Simulink to simulate the passing of entities through a network of queues, servers, gates and switches based on events. SimEvents and Simulink provide an integrated environment for modelling hybrid dynamic systems containing continuous-time, discrete-time, and discrete-event components. Here, two concepts in SimEvents need to be clearly discussed; namely Entity and Event.

- Entity: the Discrete-event simulation is typically interested in the discrete items, which are regarded as entities in SimEvents. In a simulation model, defined entities have activities in the network of queues, servers, gates and switches. Those entities can carry different contents, known in SimEvents as attributes. For example, in vehicle network, each message is an entity having attributes like data, data rate, data length and message id.

- Event: in a discrete-event simulation, an event is an instantaneous discrete incident that changes a state variable, an output and/or the occurrence of other events.

5.4.4.1 *SIMEVENTS QUEUES AND SERVERS*

SimEvents Queues

In a discrete-event simulation, a queue stores entities for some length of time that cannot be determined in advance. The queue attempts to output entities as soon as it can, but its success depends on whether the next block accepts new entities. An example of a queue is the buffer in a vehicle network, where one message stands in a line with other messages to wait for a processor to transmit it and this message cannot determine in advance how long this message must wait. The features of different queues are:

- The queue capacity, which defines how many entities the queue can hold simultaneously.
- The queue discipline, which indicates which entity departs first if the queue has multiple entities.

SimEvents Servers

In a discrete-event simulation, a server stores entities for some length of time, called the service time, and then attempts to output the entity. The features of different servers are:

- The number of entities it can serve simultaneously, which could be finite or infinite.
- Characteristics of the method of computing the service times of arriving entities.
- Whether the server permits certain arriving entities to pre-empt entities that are already stored in the server.

5.4.4.2 SIMEVENTS PATHS AND ROUTING TECHNIQUES

SimEvents Paths

An entity path is a connection from an entity output port to an entity input port, depicted as a line connecting the entity ports of two SimEvents blocks. An entity path represents the equivalence between an entity's departure from the first block and arrival at the second block. Figure 5.7 is an example of any entity that departs from the FIFO Queue block's **OUT** port while simultaneously arriving at the Single Server block's **IN** port.

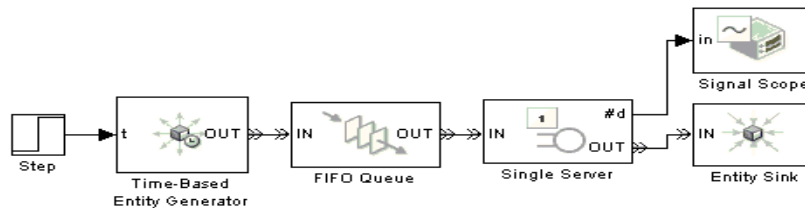


Figure 5.7: The Example of the FIFO Queue Block

SimEvents Routing Techniques

Routing techniques as a special feature in SimEvents provides a supplement to SimEvents Entities Path design. Before discussing these techniques, some SimEvents blocks have to be explained, which are shown in Table 5.2.

Block Name	Block Description	Block Diagram
Output Switch	This block selects one of the entity output ports, and the selected port can change during the simulation.	
Input Switch	This block selects one of the entity input ports. It selects exactly one entity input port for potential arrivals and makes all other entity input ports unavailable. Also, the selected entity input port can change during the simulation.	
Path Combiner	This block can merge multiple paths into a single path.	

Table 5.2: SimEvents Routing Blocks

Table 5.3 shows the Routing Techniques used in SimEvents.

Technique Name	Technique Description
Equiprobable switching	It supports the Input Switch and Output Switch blocks randomly selecting ports.
Round-robin switching	The Input Switch and Output Switch cycle through the entity input and output ports in sequence. In this technique, after the last entity port, the next selection is the first entity port.

Table 5.3: SimEvents Routing Techniques

5.6 CONCLUSION

Data transmission in an in-vehicle network is influenced significantly by the intensity and distribution of message traffic in the network. Data transmission is subject to time-varying delays due to the latency of messages. Messages may also be corrupted by noise in the network medium or lost due to buffer saturation in the receiving side. These problems increase the message transmission delay and degrade the performance of real-time control systems. For a network such as CAN, which may be shared by processors, each with a different configuration, design of an appropriate traffic load distribution is critical.

In-vehicle networks, such as CAN, LIN and FlexRay, can be modelled as a single channel queuing system with an exponential distribution for service time. The bus is the server and all waiting messages form a single queue. Therefore, it is worth using the Markov process to analyse the traffic on different buses. By using an appropriate simulation tool, the expected traffic characteristics, obtained via simulation, can be used to verify the analytical results. The analytical result may give the control system designer some idea on the relationship among message delays, message priorities and expected loads of the system and help them to choose optimum design parameters. The simulation tool can also be used as a tool to help system designers to evaluate their design.

5.7 REFERENCES

- ARSHAM, H. (1995)** Systems Simulation: The Shortest Route to Applications. 9 edition ed., National Science Founding.
- HARVEY GOULD, J. T., WOLFGANG CHRISTIAN (2006)** An Introduction to Computer Simulation Methods: Applications to Physical Systems Addison Wesley.
- IBM (1950s)** Fortran, International Business Machines Corporation (IBM).
- JACK DONGARRA, J. B., CLEVE MOLER, PETE STEWART (1980s)** LINPACK
- JACOBSEN, M. (2006)** Examples of Queuing Models Birkhäuser Boston.
- JERRY BANKS, J. S. C., BARRY L. NELSON, DAVID M. NICOL (2000)** Discrete-Event System Simulation Prentice Hall.
- Kendall, D.G. (1953)** "Stochastic Process Occurring in the Theory of Queues and Their Analysis by the Method of Imbedded Markov Chains," Annals of Mathematical Statistics, Vol. 24, pp. 338--54
- LAW, A. M., AND W.D. KELTON (2000)** Simulation Modelling and Analysis McGraw-Hill Education - Europe.
- MATHWORKS, T. (1984)** MATLAB, The MathWorks.
- MATHWORKS, T. (1997)** Simulink and Matlab User's Guides, The MathWorks, Inc.
- MATHWORKS, T. (2006)** SimEvents User's Guides, The MathWorks, Inc.
- RU, W. Q. (1995)** C Language and Computer Simulation, 北京航空航天大学出版社.
- WEISSTEIN, E. W.** Markov Process WolframMathWorld.
- WILKINSON, J. (1972-1973)** EISPACK

LITERATURE REVIEW SUMMARY

The literature review described the technologies/methodologies that the research is based upon.

Chapter Two presented the current state of in-vehicle network status with regards to the vehicle network protocols, such as CAN, TTCAN, LIN and FlexRay with their advantages and disadvantages. In this chapter, some important features and data, such as the Time-Triggered feature and Message Length, of these protocols are collected.

Chapter Three described the vehicle network design and management with traditional and advanced methods. Relate to this research, it was found that all requirements for the network should be clearly defined. Additionally, the issues and problems concerning the network, both real and perceived should be fully understood.

Chapter Four then presented the key concept of this research: vehicle network gateway with detailed discussing it from its hardware architecture and software implementation. The in-vehicle network gateway structure needs to consider several factors. The design of the gateway must be customizable, inexpensive and robust in order to maintain competitiveness.

The **final chapter** of the literature review described a reference theory which is applicable for the data transmission in the vehicle network gateway. This chapter also introduced the concept of simulation and a simulation tool was introduced to assist in the present research. In-vehicle networks can be modelled as a single channel queuing system with an exponential distribution for service time. Therefore, it is worth using queuing theory to analyse the traffic on different buses.

PART THREE

SIMULATION MODEL DEVELOPMENT

7.1 METHODOLOGY OVERVIEW

There is a new design paradigm for in-vehicle network systems called model-based design (AUTOMOTIVE ELECTRONICS, 2004), or electronic system-level (ESL) design. Virtual Prototypes are the core of the model-based design approach, which consists of software versions of the silicon systems, ECUs or networks of ECUs. Simulation is one type of virtual prototype used in automotive industry.

This research aims to develop a detailed gateway simulation model that can be used to analyze gateway performance, including the Number of Dropped Messages, Throughput and the Average Message Latency. By evaluating these performance measures, a better gateway design strategy can be achieved.

7.2 SIMULATION DESIGN

7.2.1 SIMULATION PROCESS

The simulation process consists of the following steps (BANKS, 2004).

Problem Formulation

Every simulation study begins with a problem statement. The problem statement should include a set of assumptions regarding the behaviour of the process to be simulated. Policy makers and analysts are involved in defining the problem formulation. Regardless of whether the policy makers or analysts define the problem, the others must clearly understand and agree to it.

Setting of Objectives and Overall Project Plan

This step focuses on whether simulation is a suitable methodology for the system formulated and objectives, as defined. If at this stage, simulation is thought suitable, the whole project plan might consist of a statement of alternative systems and a method for evaluating the effectiveness of these alternatives.

Model Conceptualization

This is a gradually developed step, which aims to abstract the features of the system, select and modify basic assumptions that characterise the system, whilst enriching and elaborating the model until a useful approximation of results, from simple to complex, is attained.

Data Collection

This step requires early progress during all stages of the model development. That is because as the complexity of the model changes, the required data elements may also change. There is constant interplay between the construction of the model and the collection of the input data required (SHANNON, 1975).

Model Translation

This step involves “programming” the model into a computer-recognizable format. Simulation languages, which are powerful and flexible or special-purpose simulation software such as Matlab and Simulink, should be selected to program the system model.

Verification

This step focuses on helping the simulation model developer determine if the computer program is working properly, without substantial debugging. Verification depends on if the input parameters and logical structure of the model are correctly represented in the computer.

Validation

This step decides if the correct representation of the real system is simulated by a simulation tool or language. The validation step has to be repeated to improve the model by calibrating the model, comparing the difference between the model simulated and actual system behaviour, until it is judged to be accurate and acceptable.

Experimental Design

This step focuses on making decisions on the alternatives to be simulated. For each simulated system design, the decisions should take into consideration the following factors:

- The length of the initialization period,
- The length of simulation runs,
- The number of replications to be made of each run.

Production Runs and Analysis

This step focuses on measuring the system’s design performance.

More Runs?

This step decides whether additional runs are necessary and defines additional experiments required by the design.

Documentation

This step focuses on two types of documentation: program and progress. Reasons for program documentation include:

- Reused by the same or different analysts for understanding the program operation.
- Reused by the same or a different analyst for modifying the program.
- Reused by the model users for changing parameters when deciding the input parameters that output measures of performance.

Musselman (MUSSELMAN, 1998) discusses progress reports that provide the important, written history of a simulation project.

Implementation

This step decides the result (failure or success) according to the previous steps. It is contingent upon how thoroughly the analyst has involved the ultimate model user during the entire simulation process. If the model user has been thoroughly involved and understands the nature of the model and its outputs, the likelihood of a rigorous implementation is enhanced. (PRITSKER, 1995)

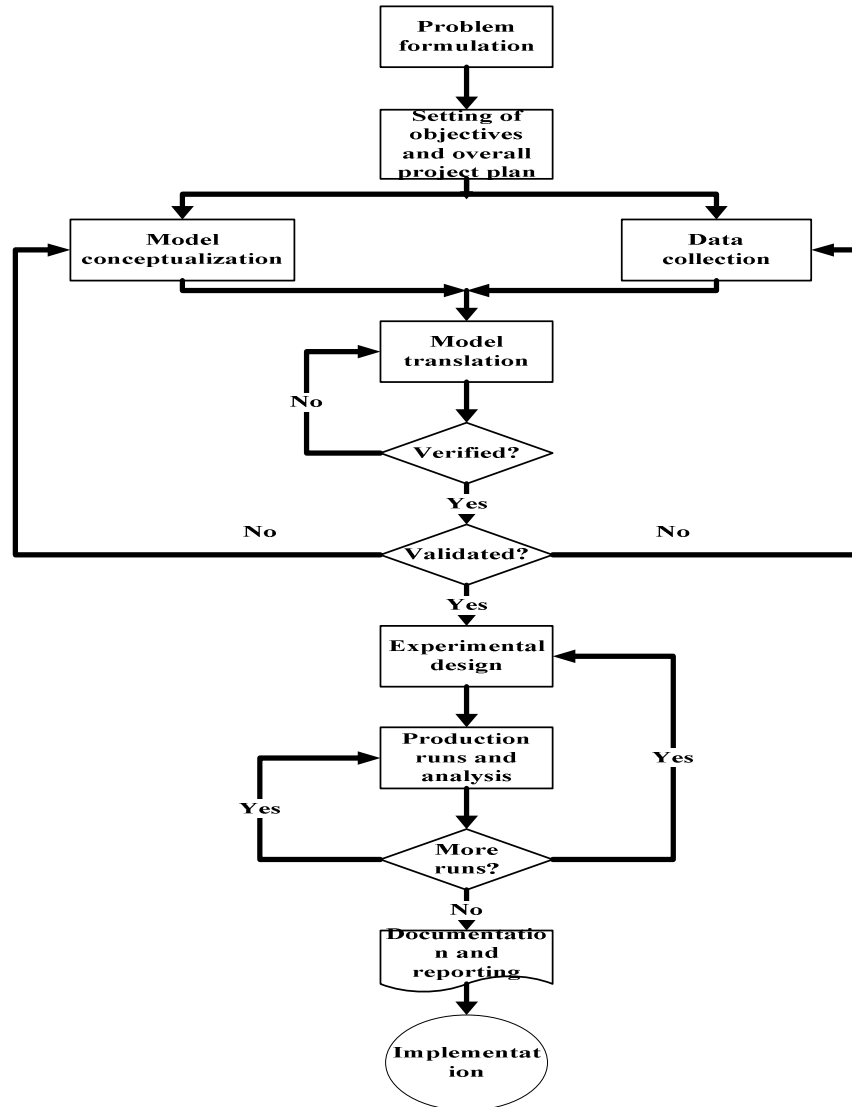


Figure 7.1: Simulation Process

7.2.2 SIMULATION PROCESS RELATED TO THIS RESEARCH

This section highlights the relationship between simulation and this research.

Problem Formulation

The current research begins with the problems of in-vehicle network gateways (Described in the Thesis Overview Chapter). Gateways in vehicle networks have become a critical factor, which is recognized by automotive industry.

Setting of objectives and Overall Project Plan

There are many ECUs and variants to consider when designing in-vehicle networks. It is difficult to build enough physical prototypes to perform adequate testing, even considering only the most critical variations. Therefore, simulating a virtual prototype of a gateway is a more effective solution for verifying its performance.

Model Conceptualization

All the basic features of in-vehicle networks are abstracted and selected to characterise the system to be simulated. These features include time-triggered and event-triggered transmission and reception.

Data Collection

All the required data elements the system needs are collected and analyzed.

Model Translation

It was decided that the Matlab/Simulink and SimEvents would be used for the model simulation, because it is widely used in the automotive industry.

Experimental Design

The in-vehicle network model will initially be simulated as different network components. These components are:

- Protocol controllers: event and time triggered transmission and reception.
- Communication Bus: Bus Loading, Bus Rate and Time Triggered Communication.
- In-vehicle Network Gateway: Signal Gateway and PDU Router according to the AUTOSAR defined specification.

Verification

The in-vehicle network simulation model will be verified using Matlab/SimuLink and SimEvents to make sure the model performs properly.

Validation

The verified in-vehicle network model will be sent to an expert in automotive industry for validation. Upon receiving the comments from the expert, the simulation model will be refined until model accuracy is judged acceptable. In the final stage of this validation process, a number of gateway model evaluation sets will be setup to optimise the gateway performance. Some important parameters and expected results will be chosen to implement this validation process. This step will be implemented by using a three-step approach, which is described in the Testing Chapter, formulated by (NAYLOR,1966).

7.3 *SIMULATION TOOL SELECTION*

There are many features that are relevant when selecting simulation software (BANKS, 2000A). The following advice can be applied when evaluating and selecting simulation tools:

1. There are a number of issues that need to be considered:
 - The accuracy and level of detail obtainable,
 - Ease of learning,
 - Vendor support,
 - Applicability to the problems.
2. The selected tool's execution speed is important. Speed affects development time. During debugging, the analyst may have to wait for the model to approach the point in simulation time where an error occurs many times before the error is identified.
3. The advertising claims and demonstrations of the simulation tool the selected should be read carefully, simply because many advertisements only exploit the positive features of the software. Similarly, the demonstrations solve the test problem very well, but perhaps not the problem.
4. Always contact the vendor, even to solve apparently insignificant problems.
5. Beware of "checklists" with "yes" and "no" as entries. For instance, many packages claim to have a conveyor entity; however, implementation and capability are what are important.
6. Many Simulation tools provide a feature which can link the simulation model to, and use code or routines written in external languages such as C, C++, or VB. This is a good feature, especially when the external routines already exist and are suitable for the purpose at hand. However, the more important question is whether the simulation package and language are sufficiently powerful to avoid having to write logic in any external language.

There may be significant trade-offs between the graphical model-building environments and those based on a simulation language. While graphical model building removes the learning curve due to language syntax, it does not remove the need for procedural logic in most real-world models and the debugging to get it right. Beware of “no programming required” unless either the package is a near-perfect fit to your problem domain, or programming is possible with the supplied blocks, nodes, or process flow diagram, in which case “no programming required” refers to syntax only and not the development of procedural logic.

At the beginning of the simulation tool selection, a simulation tool called SIMUL8 was used to model the in-vehicle network and gateway. However, it was determined that it is quite hard to gather all the data requirements using SIMUL8 to build the model. There are no existing examples and references on the vehicle network simulation using this tool. It appears that SIMUL8 cannot be applicable to the automotive industry.

Matlab/Simulink, however, is a well know simulation tool in the automotive industry. The new SimEvents package provides additional features such as the Events Generate, Queues and Servers and Paths and Routing Techniques which are applicable to the model simulated in this research.

7.4 CONCLUSION

Today, the number of functions in a vehicle ECU is increasing, which in turn is causing the number of ECUs in vehicles to increase. The more ECUs that are connected onto networks, the more burden a gateway will have. Many OEMs have to consider lots of variations when designing vehicle gateways. In order to design a reliable gateway, those variations must be accounted for during testing. Using the prototyping approach it is quite hard to build enough physical prototypes to perform adequate testing, even with the most critical variations. Simulating a virtual prototype of a gateway is a more effective solution for verifying its performance. It is for this reason that Matlab and Simulink were used in the research.

7.5 REFERENCES

- AUTOMOTIVE ELECTRONICS, V. (2004)** Automotive Electronics: Model-Based Development with Virtual Prototypes. VaST System Technology.
- JERRY BANKS, J. S. C., BARRY L. NELSON, DAVID M. NICOL (2000)** Discrete-Event System Simulation Prentice Hall.
- MUSSELMAN, K. J. (1998)** Guidelines for Success, John Wiley & Sons Inc.
- NAYLOR, T. H., J.L.BALINFY, D.S.BURDICK, AND K.CHU (1966)** Computer Simulation Techniques, John Wiley, New York.
- PRITSKER, A. A. B. (1995)** Introduction to Simulation and Slam II John Wiley & Sons Inc.
- SHANNON, R. E. (1975)** Systems Simulation: The Art and Science, Prentice Hall

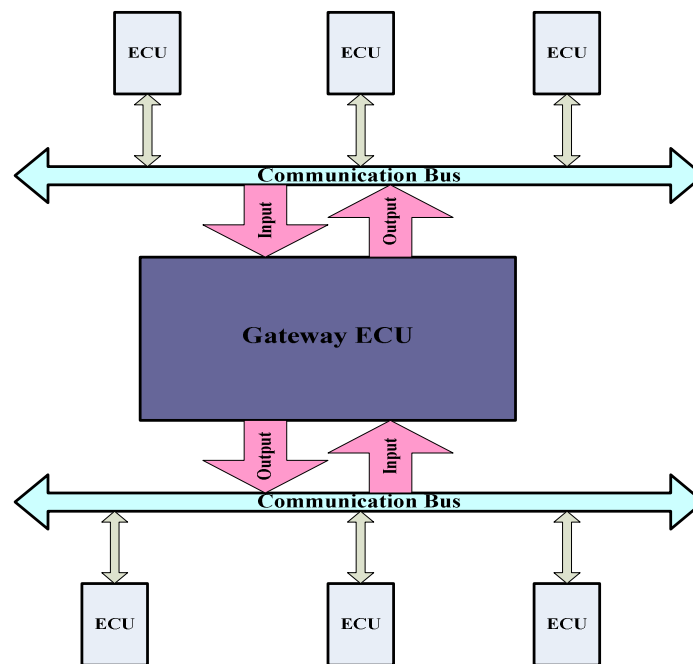
**GATEWAY MODEL
REQUIREMENTS SPECIFICATION****8.1 SYSTEM ARCHITECTURE OVERVIEW****Figure 8.1: System Architecture**

Figure 8.1 shows an overview of a simple in-vehicle network system and its components, including the gateway.

- ECUs (Electronic Control Units): control a variety of vehicle functions, including fuel injection, transmission shifting and anti-lock braking.
- Communication Buses: are the network(s) used in the vehicles, for example CAN, LIN, FlexRay, etc.
- Gateway ECU: is connected to two or more different networks and controls the information interchange between these networks.

8.2 AUTOSAR GATEWAY STRUCTURE

AUTOSAR (AUTOSAR, 2006) has recently released a detailed document on in-vehicle network gateways. Figure 8.2 shows the basic gateway structure defined by AUTOSAR.

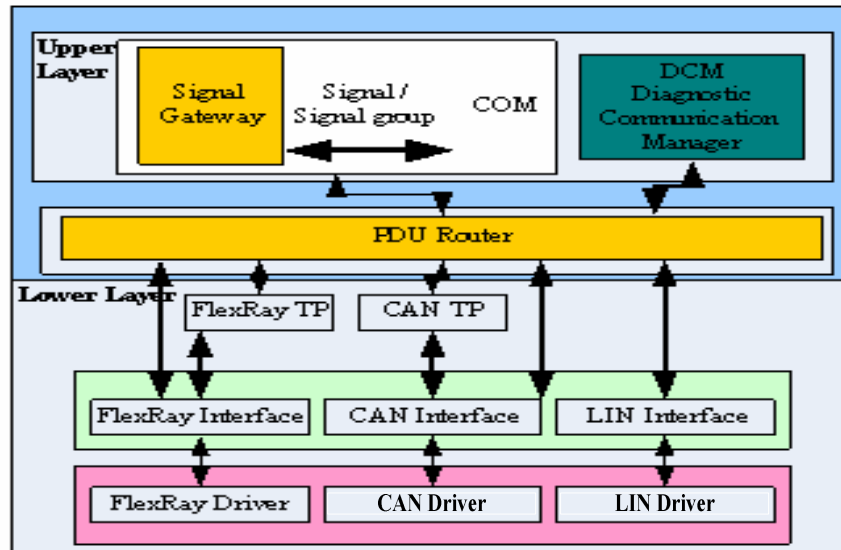


Figure 8.2: AUTOSAR gateway structure

COM is a method of exchanging data between different tasks and between multiple ECUs over a network. COM is an asynchronous communication model, where the application is not required to wait for a message transfer before it resumes processing and it is not blocked if a message is not available when requested. COM defines a number of notification mechanisms that assist the application in understanding when a message is sent or received.

Each message defined for an application can have only one sender within the system, but one or many receivers can receive it. These receivers can be tasks that reside on the same or a different ECU.

Signal-based Gateway is situated inside of the COM. Signals and signal groups are stamped with unique static names by signal router. A routing table mechanism is used to store all the signals or signal groups' information including their names, destination, etc. Its main tasks are:

- Packing signals or signal groups (Complex Data Types) from applications into message units (PDU).
- Unpacking signals or signal groups from message units (PDU) to applications.

The **PDU Router** is located between Upper Layer COM and Lower Layer. Its main tasks are:

- Providing a transport platform for messages with different protocol formats on the Lower Layer.
- Providing a transport platform for different functional networks on the Lower Layer.
- Providing a transport platform for packing/unpacking messages into the Upper Layer.

The **Lower Layer** includes Transport Protocols, Protocol Interface and Protocol Drivers, which are close to the physical layer.

8.2.1 MESSAGE TRANSMISSION TYPES

According to the AUTOSAR defined Gateway specification, the transmit side of a controller have three transmission types:

- Non TP-PDU-TX without Trigger Transmit: data to be transmitted in this case is required to send a transmit request first. Data will be transmitted on the related bus after that transmit request, without delay. After successful transmission, a transmit confirmation is provided.

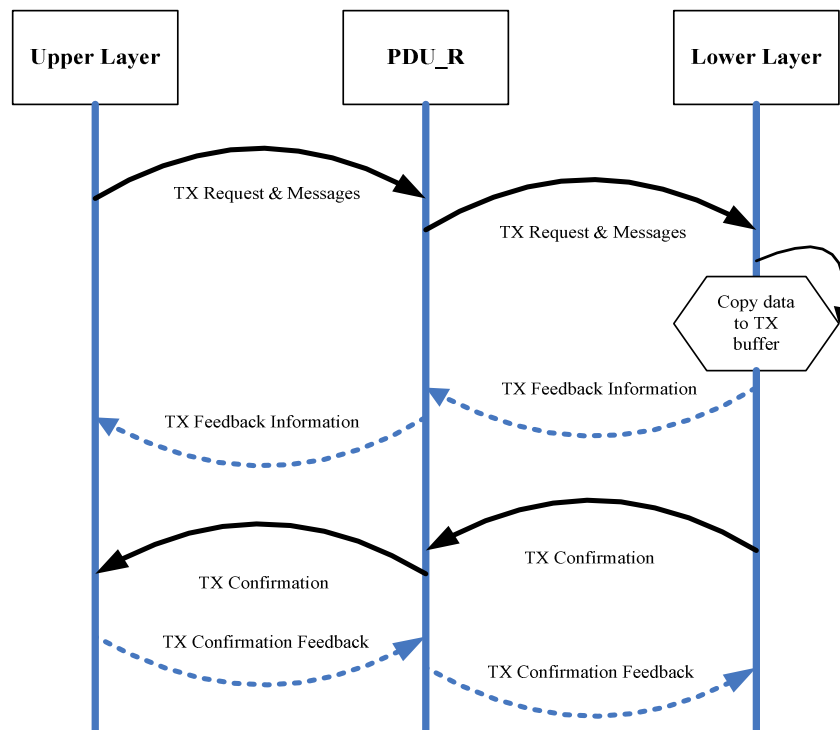


Figure 8.3: Non TP-PDU-TX without Triggered

- Non TP-PDU-TX with Trigger Transmit: data will not be transmitted right after a transmit request. Data transmission is implemented by a Trigger Transmit signal from the gateway. In this case, a buffer is provided by the gateway ECU.

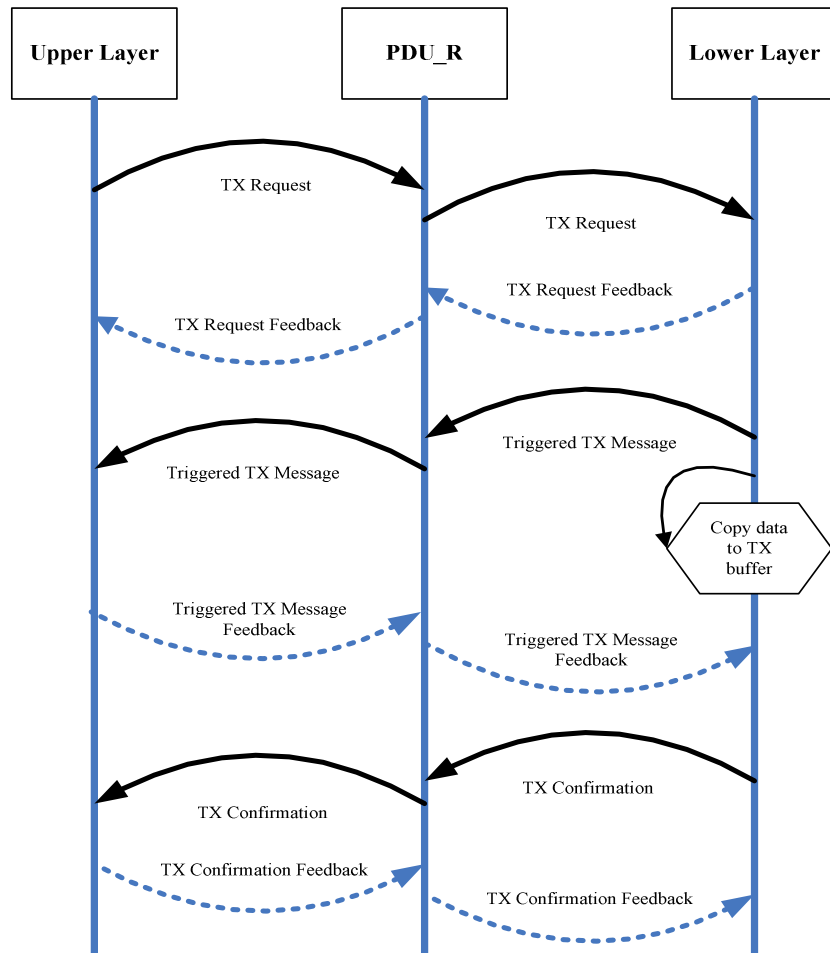
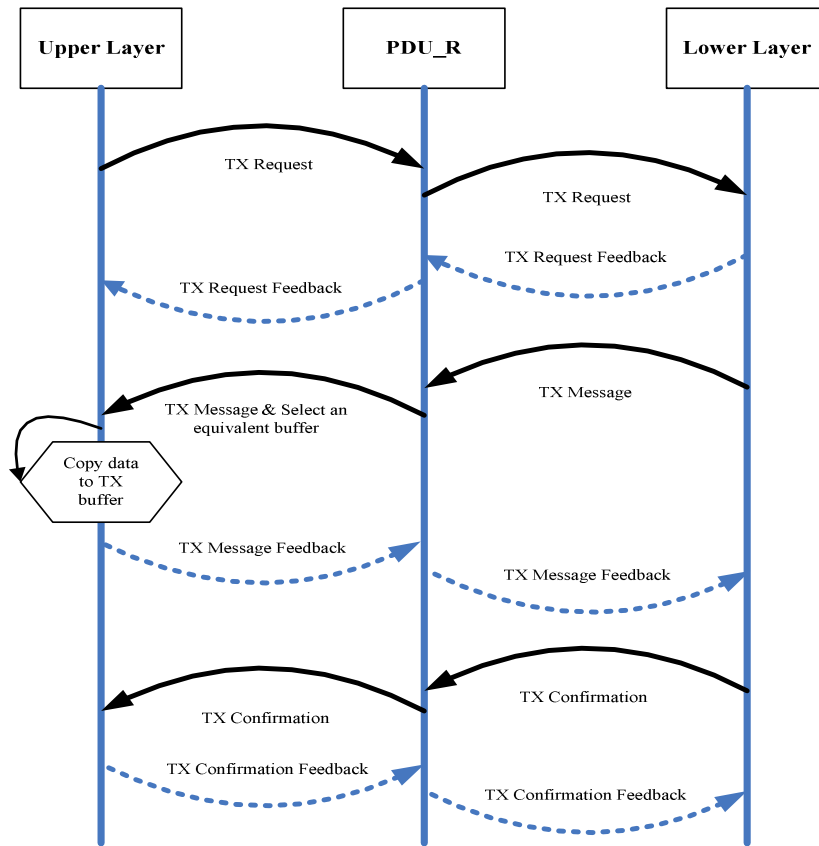


Figure 8.4: Non TP-PDU-TX with Trigger Transmit

- TP-PDU-TX: data transmission happens via the transport protocol modules. The transmit request is forwarded by the gateway ECU to the related TP module. Depending on each message length, an equivalent transmit buffer is selected. The TP module will transmit the message from the transmit buffer. For an efficient usage of the transmit buffer, the transmit side should be configured for multi-sized buffers. After successful transmission, a transmit confirmation is provided.

**Figure 8.5: TP-PDU-TX**

8.2.2 MESSAGE RECEPTION TYPES

According to the AUTOSAR defined gateway specification, each receive operation of the receive side in a controller shall always be triggered by an indication. The indication is either invoked by an interrupt or results from polling another communication controller. There are two types of message reception.

- Non TP-PDU-RX: data to be received is not from the transport protocol module but from an interface module.

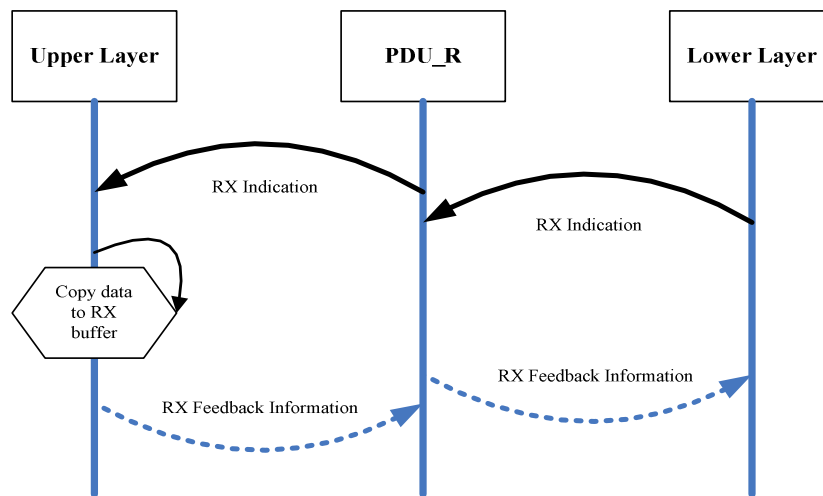


Figure 8.6: Non TP-PDU-RX

- TP-PDU-RX: data to be received is from a transport protocol module. Depending on each message length, an equivalent receive buffer is selected. For efficient usage of the receive buffer, the receive side should be configured for multi-sized buffers.

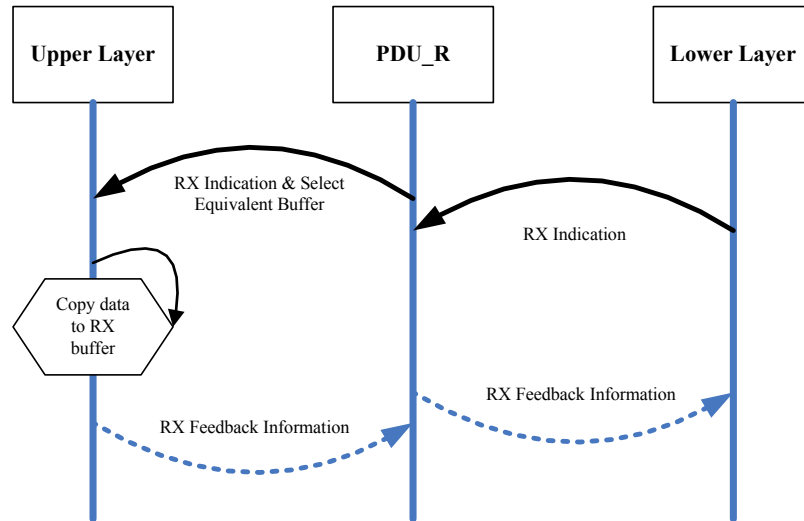


Figure 8.7: TP-PDU-RX

Each receive side should be configured by a message filter. The message filter works on the principle of a front door, that is, deciding which message should be accepted by checking each message's identifier number.

8.2.3 AUTOSAR PDU ROUTER

The ECU can act as a direct gateway between two interface modules without rate conversion. It can also act as a non-direct gateway between two interface modules with rate conversion and two transport protocol modules. There are three types of gateway.

- Non TP-PDU-Gateway without/with Rate Conversion: this acts as a direct gateway between two interface modules. Messages received from one interface shall be forwarded to the other interface. In this case, the TP-PDU-Gateway contains both triggered and non-triggered gateway transmit operations. The triggered idea can be referred to the messages transmission part, discussed in the early this chapter.

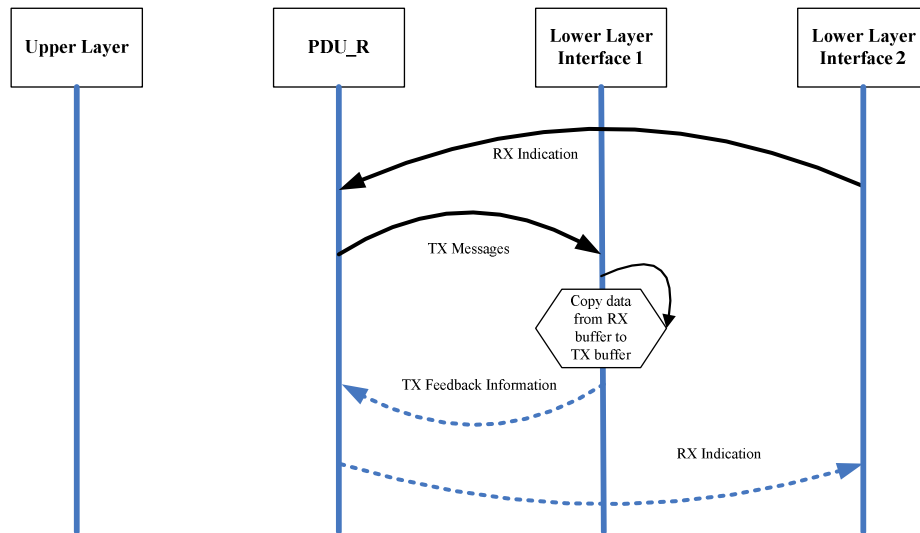


Figure 8.8: Non TP-PDU-Gateway without Rate Conversion (Non-Triggered)

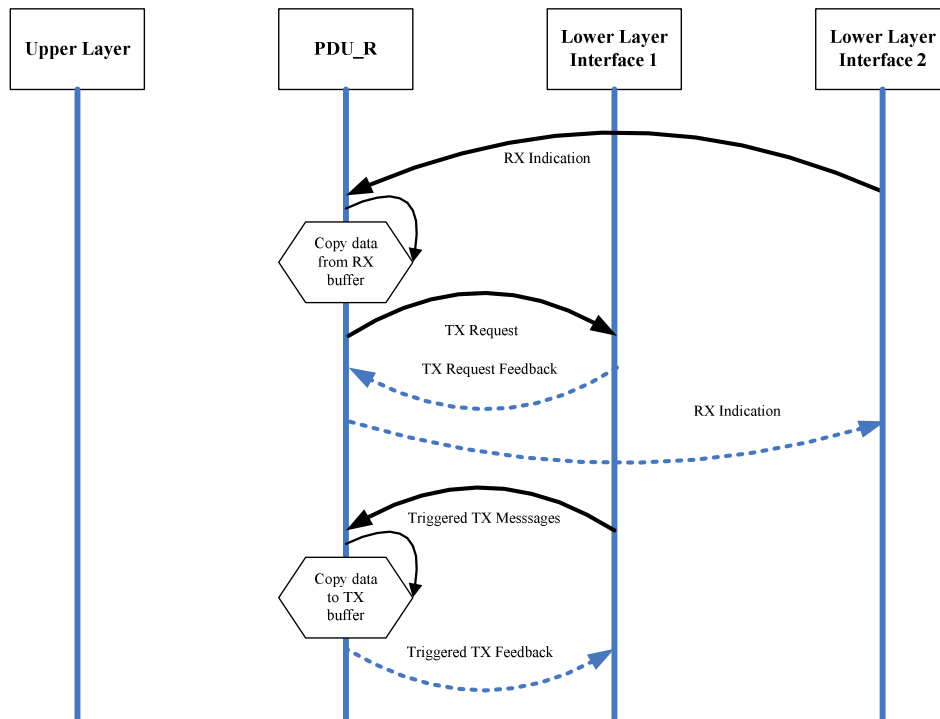


Figure 8.9: Non TP-PDU-Gateway without Rate Conversion (Triggered)

- Non TP-PDU-Gateway with Rate Conversion: the PDU Router does not directly support communication between two interface modules with rate conversion. Rather, the rate conversion is support by the Upper Layer of the gateway ECU; the signal gateway in the Upper Layer carries out the rate conversion.

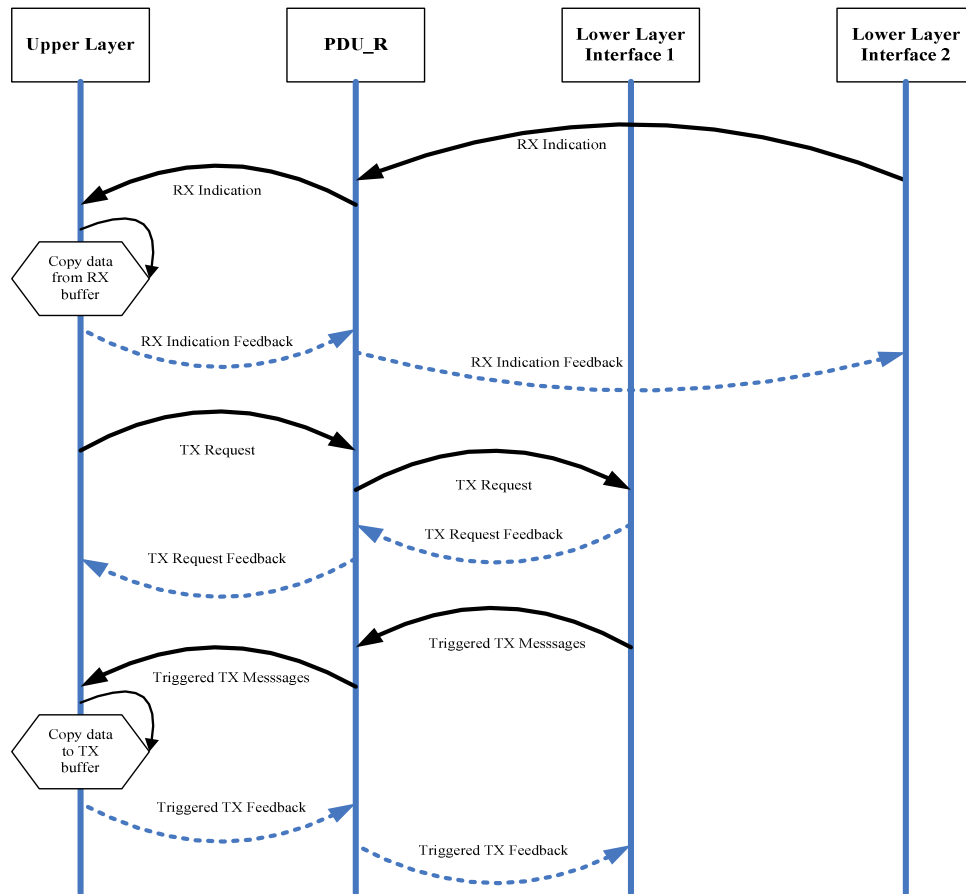


Figure 8.10: Non TP-PDU-Gateway with Rate Conversion

- TP-PDU-Gateway: data received from one bus shall be forwarded to another bus. In this case, it consists of two parts: TP PDU reception and TP PDU transmission, both via transport protocol modules. The PDU Router shall support routing on-the-fly. Therefore data transmission and reception on both transmit and receive TP modules must happen simultaneously.

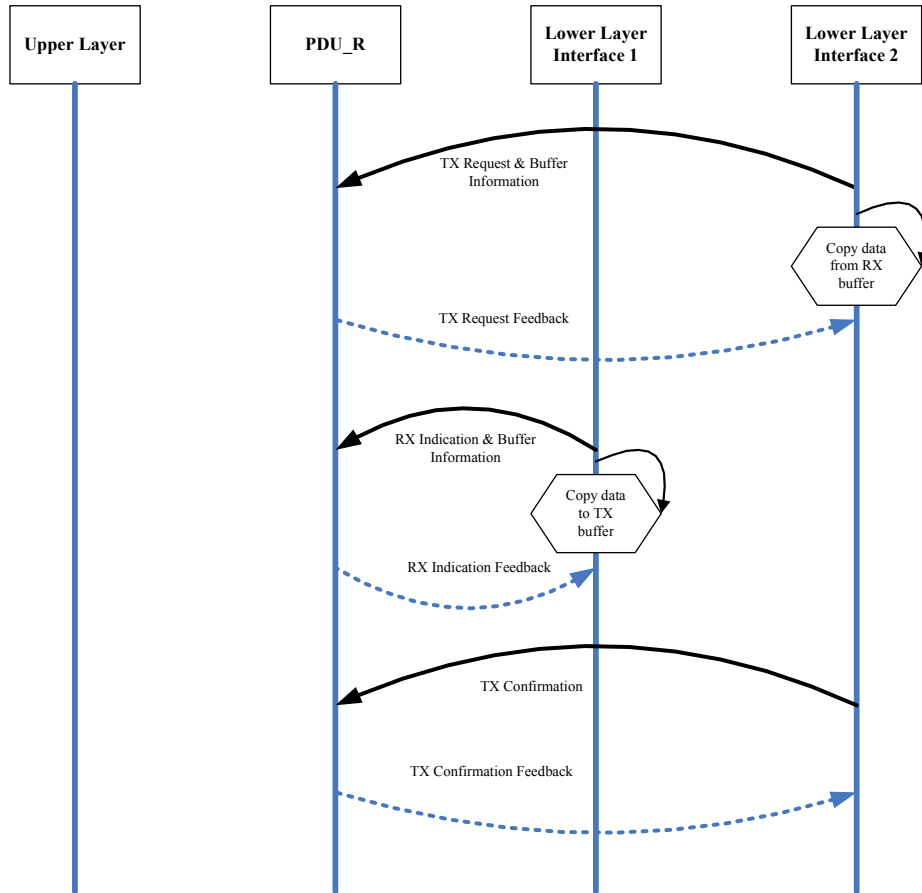


Figure 8.11: Gateway Block

8.3 *GATEWAY BUFFER REQUIREMENTS*

In the case of a gateway, transmission of each message routed between different interface modules shall have a configured buffer in the PDU Router. Different buffer sizes are possible. There shall be, if configured, a separate buffer for messages with differing length.

The buffers in the PDU Router can be configured using the following parameters:

- Buffer Length: length of buffers for each PDU router can vary from 1 to n bytes.
- Buffer Overwrite: if the buffer is full, the buffer shall be flushed and the new value shall be forwarded to the interface.
- Trigger Transmit in case of Empty Buffer: if the interface requests a value but the PDU router buffer is empty the most recent value shall be provided. If there has been no transmission before, the most recent value shall be the default value.

The PDU Router specifies the strategy, and configures each message for routing. A PDU can also be configured to contain no buffer.

8.4 *CONCLUSION*

The purpose of this chapter is to provide a clear understanding of the PDU Router. Those sequence diagrams which illustrated different communication scenarios in this chapter are discussed further in chapter nine.

8.5 *REFERENCES*

AUTOSAR (2006) SPECIFICATION OF PDU ROUTER, AUTOSAR GbR.

SIMULATION MODEL DESIGN AND IMPLEMENTATION

9.1 INTRODUCTION

The in-vehicle network gateway system was prototyped and tested using the MathWorks MATLAB/Simulink and SimEvents (MATHWORKS, 1997B). The overall system was designed based on the AUTOSAR Gateway specification (AUTOSAR, 2006). Firstly, the separate components have to be prototyped. Three main components of the system were prototyped:

- Simulating different types of protocol interfaces (CAN, LIN and FlexRay) with the ability to send and receive messages and the transmission types: time-triggered and event-triggered.
- Simulating the communication bus that would communicate with different protocol interfaces.
- Simulating AUTOSAR defined gateway structure, which should have the ability to connect different protocol interfaces and communication bus simulations.

Figure 9.1 shows the whole in-vehicle network system to be simulated. The system is composed of two networks, each with three ECUs and featuring time-triggered and event triggered communication. A gateway ECU integrates a signal gateway and a PDU router. The detailed simulation content is described in the following subsections.

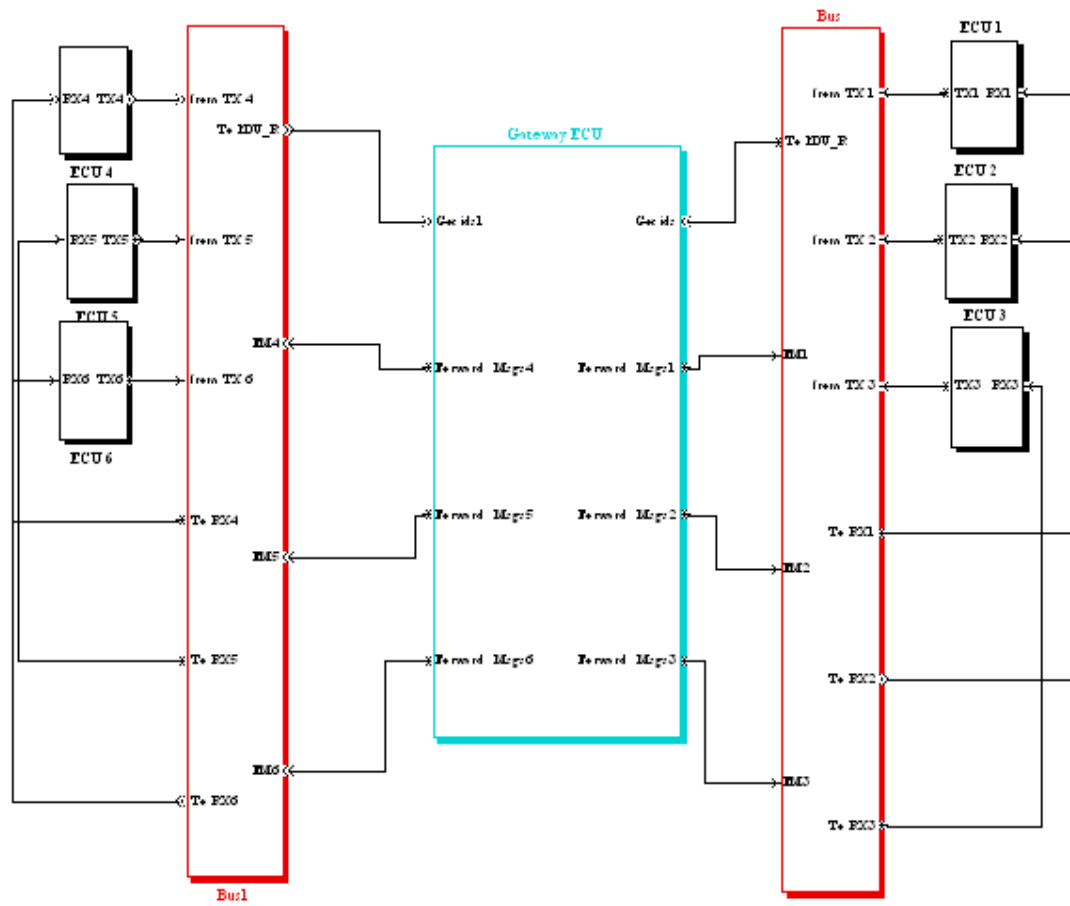


Figure 9.1: In-vehicle Network Simulation Model

9.2 REST BUS SIMULATION MODEL

The simulation of different protocol interfaces could have two types: Event-triggered and Time-triggered. Due to the system design requirements, only the features of the transceivers related to these requirements were considered, such as buffers, message packing and unpacking.

For the simulation design, it was considered that CAN should be the main protocol for simulation, as CAN contains event-triggered and time-triggered features (for example TTCAN).

CAN Block Functionality

To judge in-vehicle network gateway efficiency, it is essential to prototype some devices. Therefore, some devices were represented as a CAN message request sent to and received from the Gateway ECU over the communication bus.

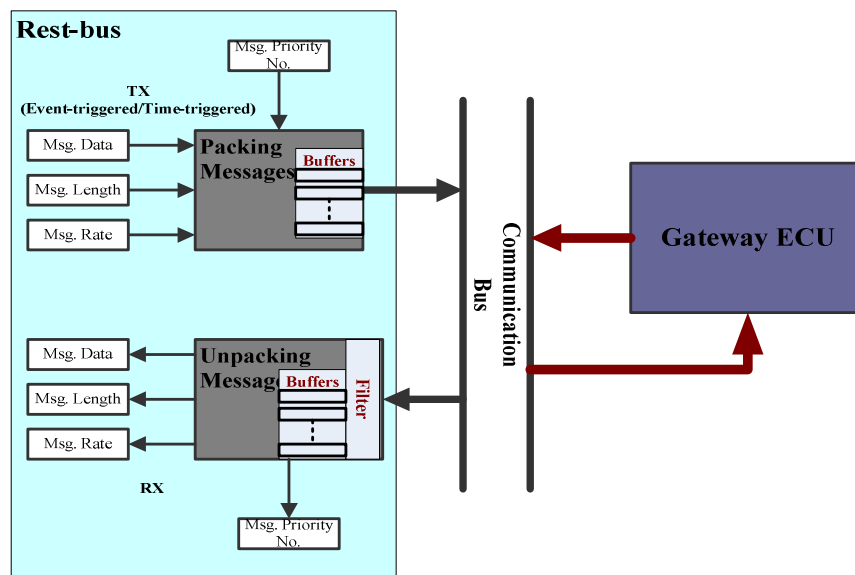


Figure 9.2: Rest Bus Block

Gateway ECU is the main component to be simulated, therefore only the major features, as shown in Figure 9.2, were considered. These include: Packing/unpacking messages. Each message should consist of Msg. Priority No., Msg. Data, Msg. Length and Msg. Rate. Table 9.1 contains the definition for each message attribute.

Msg. Priority No.	Indicating the queuing sequence in the buffer
Msg. Data	Containing the data of each message
Msg. Length	Indicating the length of each message to be transmitted. This attribute can be used for each message to select the buffer
Msg. Rate	Indicating the transmission rate of each message. This attribute can be used for each message, if this message needs to be converted to a different rate

Table 9.1: Message Attribute definitions

9.2.1 COMMUNICATION BUS SIMULATION MODEL

The communication bus works as a “super highway” between different networks and gateways. Therefore, to monitor the whole network system, the communication bus is a critical factor.

For a gateway design, some communication bus features should be considered.

- **Bus Rate:** as discussed previously, different networks can have different data transfer speeds.
- **Bus Load:** shows the percentage of the bus capacity being used. It is important, because the gateway will control the network traffic based on the bus load. For example, if the bus load is quite high, the gateway will manage the messages to be transmitted/received under a low frequency, and if the bus load is quite low, the gateway will manage the messages to be transmitted/received under a high frequency.
- **Message Transmission Time:** makes an in-vehicle network system more realistic to the real world. Each message is assigned a time, indicating how long a single message takes to travel across a communication bus. This transmission time depends on Bus Rate, Bus Load and Message Length. The transmission times were derived experimentally for various combinations of Bus Rate and Bus Load. These were then added to the bus simulation model.

9.2.2 EXPERIMENTAL MEASUREMENT OF MESSAGE TRANSMISSION TIMES

The message transmission time is the time that each single message is transmitted onto the bus. To determine this time, a hardware device is required. By writing a small piece of C code, as shown in Figure 9.3, the signal message transmission time can be monitored from the timer of the Infineon C166® (INFINEON) board. Table 9.2

shows how to set up the Infineon hardware. In this transmit program, a single message is transmitted ten times to the bus during each transmit cycle, which can get the timing of the transmit from the Timer more accurately. The timer is stopped after transmission of ten messages by setting a break point in the debugger.

```

void main(void)
{
    MAIN_vInit();

    while (1)
    {
        GPT1_vClearTmr(GPT1_TIMER_3); // Clear Timer once one transmit cycle finished.
        GPT1_vStartTmr(GPT1_TIMER_3); // Before another transmit cycle, Timer has to be started.
        CAN1_vTransmit(1);           // Call the CAN transmit function to start a transmit cycle.

        while(1);
    }
} // End of function main

interrupt (T3INT) void GPT1_viTmr3(void)
{
} // End of function GPT1_viTmr3

interrupt (XP0INT) void CAN1_viCAN1(void)
{
    uword uwIntID;

    while (uwIntID = C1PCIR & 0x00ff)
    {
        switch (uwIntID & 0x00ff)
        {

            case 3: // Message Object 1 Interrupt

                CAN1_OBJ[0].MCR = 0xfffd; // reset INTPND

                // The transmission of the last message object
                // was successful.

                GPT1_vStopTmr(GPT1_TIMER_3); // Once one message is transmitted on to the bus, Stopping the Timer.
                Check_TX_No();               // Check how many messages have been transmitted.
                GPT1_vClearTmr(GPT1_TIMER_3); // Clear the Timer
                GPT1_vStartTmr(GPT1_TIMER_3); // Start Timer again for another transmit cycle.
                CAN1_vTransmit(1);           // Call the CAN transmit function.

                break;

            default:
                break;

        } // end switch()

    } // end while

}

void CAN1_vTransmit(ubyte ubObjNr)
{
    CAN1_OBJ[ubObjNr - 1].MCR = 0xe7ff; // set TXRQ,reset CPUUPD
}

void Check_TX_No(void) // This function checks how many times a single message has been transmitted,
// once the counter i equal to 10, the whole process will start again.
{
    j = j + T3;
    i ++;

    if (i == 10)
    {
        i = 0;
        j = 0;
    }
}

```

Figure 9.3: Single Message Transmission Code

Function	Function Name	Function Explain
Timer	GPT1_vClearTmr (GPT1_TIMER_3)	This macro stops the selected GPT1 timer and sets the timer register to 0.
	GPT1_vStartTmr (GPT1_TIMER_3)	This macro starts the selected GPT1 timer. The timer continues to count from where it had stopped.
	GPT1_vStopTmr (GPT1_TIMER_3)	This macro stops the selected GPT1 timer. The contents of the timer register remain unchanged.
Transmit	CAN1_vTransmit(1)	This function triggers the CAN1 controller to send the selected message. If the selected message object is a TRANSMIT OBJECT then this function triggers the sending of a data frame. If however the selected message object is a RECEIVE OBJECT this function triggers the sending of a remote frame.
Transmit Times	Check_TX_No()	This function counts how many times one single message has been transmitted. The initial number is setup to 10.

Table 9.2: Function Description

Collecting these times from the Timer, as shown in Table 9.3, shows each bus load corresponding to its related simulation time and Msg/Sec. In this table, the value of Msg/Sec is setup and monitored from the PCAN (PEAK) software, described in the Bus Load Simulation section. The value of Msg Tim is collected from the device timer, in this instance, because the same message is transmitted ten times in one transmit cycle, thus, the Msg Tim is the result of the device timer divided by ten. The Value of the Sim Time is the Msg Tim multiplied by the device timer prescaler.

[illegible]

Table 9.3: Messages Transmission Parameter Details

9.2.3 BUS LOAD SIMULATION

It is essential to apply some industry standard tools to assist with in-vehicle network design, so that the designed model would be more applicable to other engineers.

“The PCAN Explore is a Windows-based universal monitor for control of the data stream on a CAN network. The PCAN Explorer shows the data traffic in a CAN network quickly and clearly.” (PEAK)_By generating the network traffic, the bus rate and bus load will be displayed for monitoring.

PCAN nets allow the connection of a PCAN program (client) to the CAN hardware. Miscellaneous nets can be defined in order to allow the installed hardware to operate with different Bus rates. Figure 9.5 and 9.6 show an example of a bus rate of a 500 Kbits/Sec CAN network. Ten transmit messages are generated on this network. By changing transmit period of each message, the parameter of Msg/Sec can be monitored in the PCAN Status Display window. In this example, lower and higher ids are used to make message transmission more realistic. Therefore, the bus load percentage and the parameter of Msg/Sec can be manually matched using a formula, as shown in Figure 9.4:

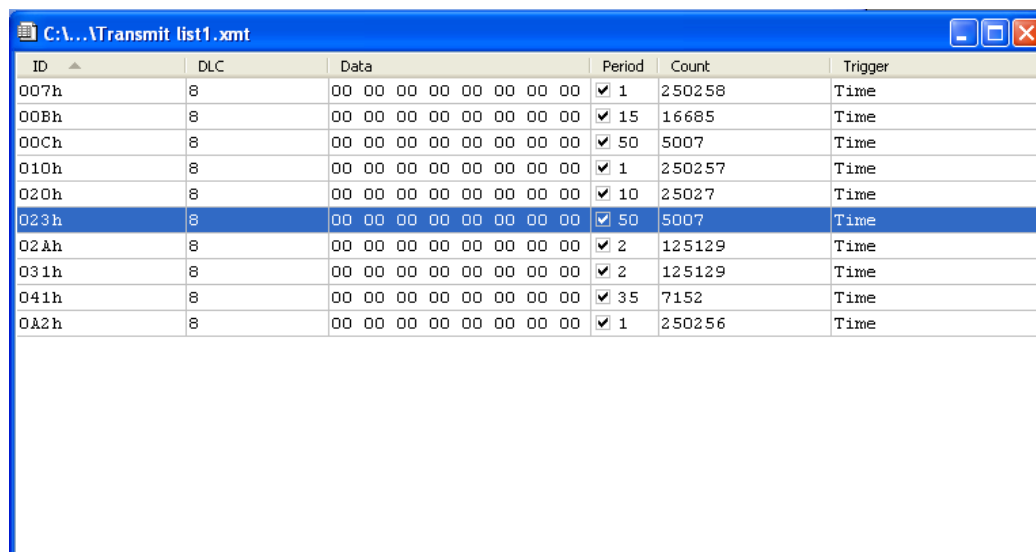
$$\text{Msg/Sec} = \frac{\text{Bus Rate} * \text{Bus Load} * 0.01}{\text{Maximum Message Length}}$$

Figure 9.4: Formula for Messages per Second

In the above formula, the length of a CAN message varies depending on the number of data bytes transmitted. Table 9.4 shows the effective length of a CAN message as a function of the data field length.

Number of data bytes	0	1	2	3	4	5	6	7	8
Minimum message length	44	52	60	68	76	84	92	100	108
Maximum message length	51	60	70	80	89	99	108	118	128

Table 9.4: Effective Length of CAN message



The screenshot shows a window titled "C:\...\VTransmit list1.xmt". It contains a table with the following columns: ID, DLC, Data, Period, Count, and Trigger. The data rows are as follows:

ID	DLC	Data	Period	Count	Trigger
007h	8	00 00 00 00 00 00 00 00	✓ 1	250258	Time
00Bh	8	00 00 00 00 00 00 00 00	✓ 15	16685	Time
00Ch	8	00 00 00 00 00 00 00 00	✓ 50	5007	Time
010h	8	00 00 00 00 00 00 00 00	✓ 1	250257	Time
020h	8	00 00 00 00 00 00 00 00	✓ 10	25027	Time
023h	8	00 00 00 00 00 00 00 00	✓ 50	5007	Time
02Ah	8	00 00 00 00 00 00 00 00	✓ 2	125129	Time
031h	8	00 00 00 00 00 00 00 00	✓ 2	125129	Time
041h	8	00 00 00 00 00 00 00 00	✓ 35	7152	Time
0A2h	8	00 00 00 00 00 00 00 00	✓ 1	250256	Time

Figure 9.5: Message transmission details

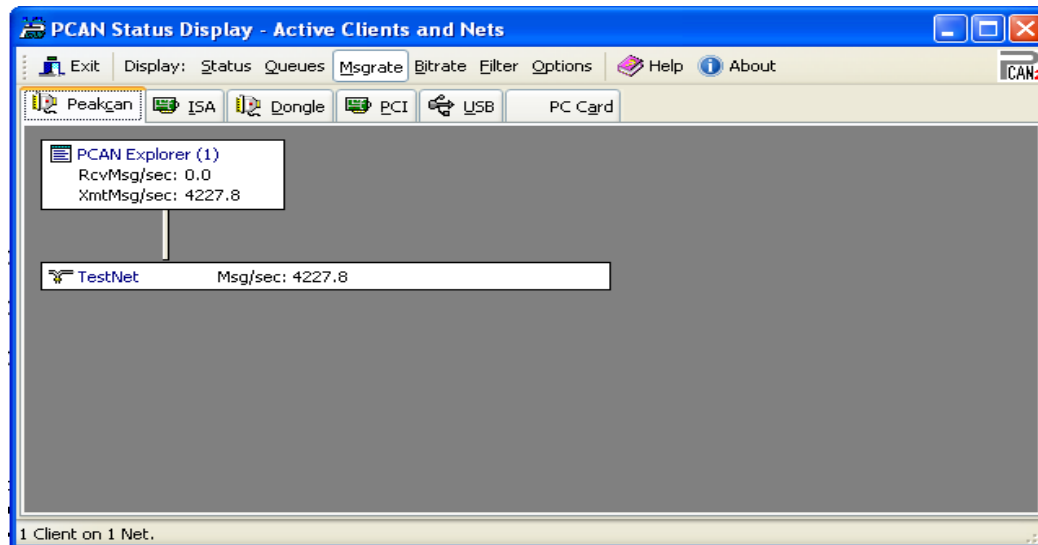


Figure 9.6: Network Status Display

Figure 9.7 shows the Bus Load for 250 Kbits/Sec simulation converted from a virtual network generated by PCAN. The Average Msg Arrival Rate slip bar defines the message range of the 250 Kbits/Sec bus load.

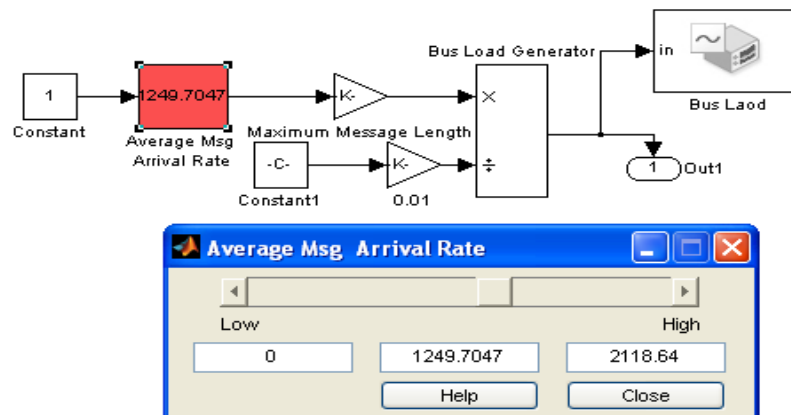


Figure 9.7: Bus Load for 250 Kbits/Sec Simulations

9.2.4 MESSAGE TRANSMISSION TIME SIMULATION

An equivalent simulation model is displayed in Figure 9.8. Some simulation time S-Function blocks are programmed for different CAN bus rates.

The output from the Bus Load for 250Kbit/Sec is the percentage of the bus usage, and this figure goes to the S-Function Builder (explained in section 9.3.1) blocks SimTime

for 250Kbit/Sec. The SimTime for the 250Kbit/Sec block converts the inputs to the outputs of the simulation time.

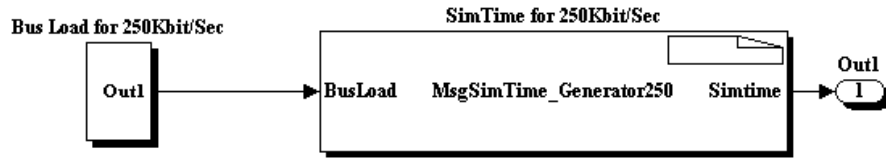


Figure 9.8: S-Function Builder for simulation time

9.2.5 TIME-TRIGGERED BUS SIMULATION MODEL

For the time-triggered bus simulation, a time scheduled message subsystem needs to be created. Each individual message to be transmitted is given a time slot by a sequence of time windows following the reference message. Figure 9.9 shows this subsystem in the Bus subsystem. In this figure, each Time-Based Message Generator block periodically generates a message in a certain predefined time, the Set Attribute block then assigns an identifier for this message. Table 9.5 shows these predefined times and identifiers.

Message Identifier	Message Period
100	10
101	20
102	30
103	40
104	50
105	60
106	70
107	80

Table 9.5: Time Schedule Message Identifiers and Periods

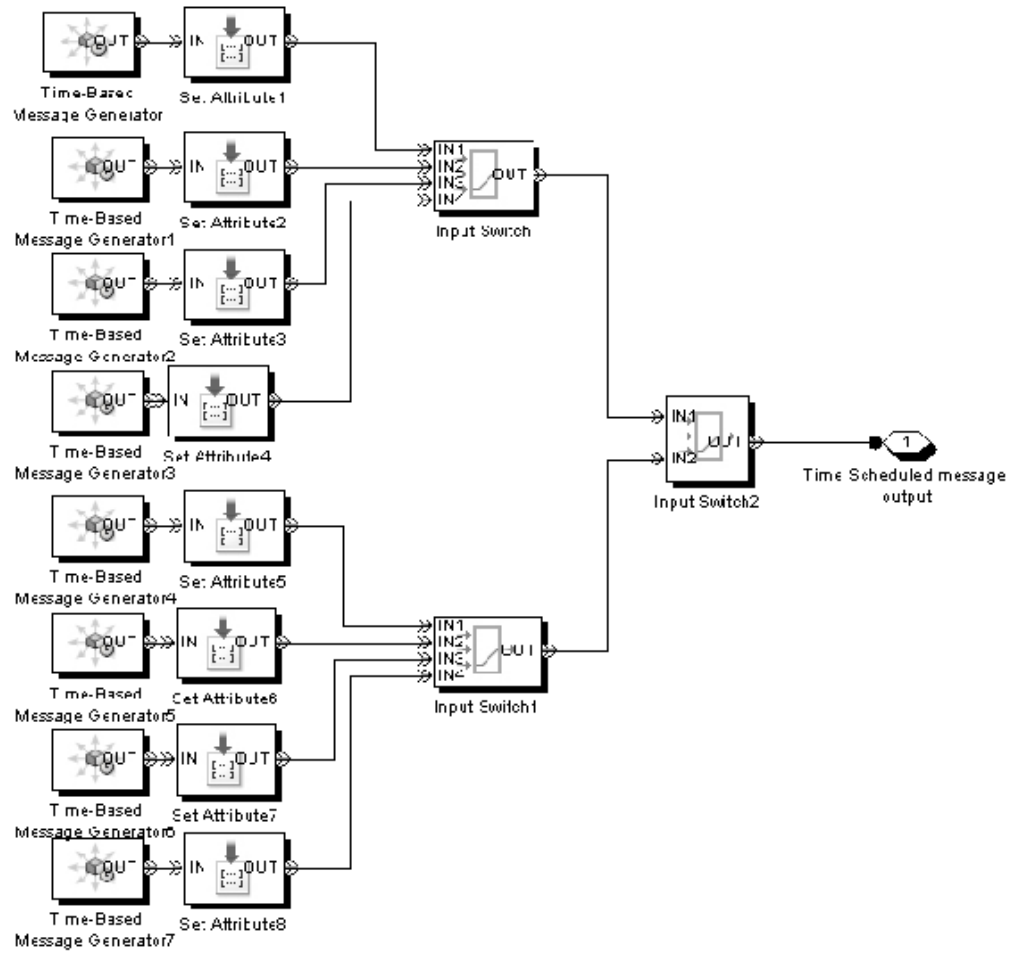


Figure 9.9: Time Scheduled Messages Transmission

9.3 AUTOSAR GATEWAY SIMULATION MODEL

The gateway simulation model is the critical part in this whole in-vehicle network model. Figure 9.10 displays two main components in this gateway model: COM and PDU Router. Details of this model are described in more detail later in this section.

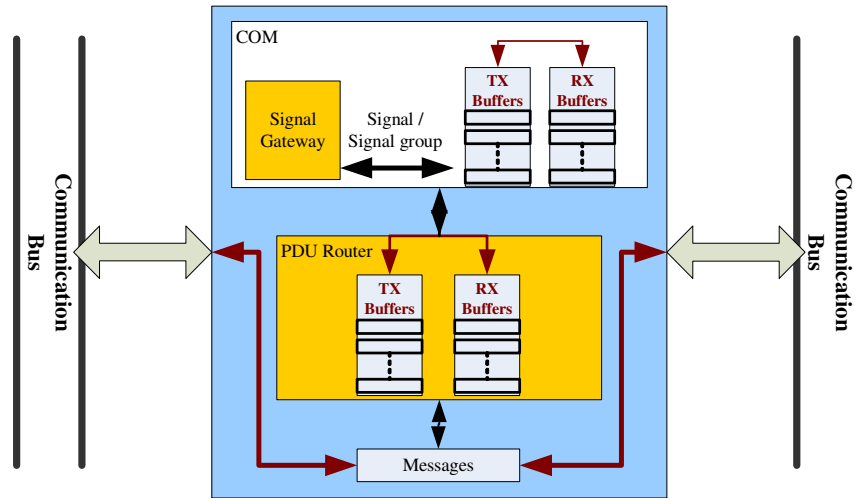


Figure 9.10: Gateway Block

Figure 9.11 shows the structure created by the Matlab/Simulink and SimEvents packages.

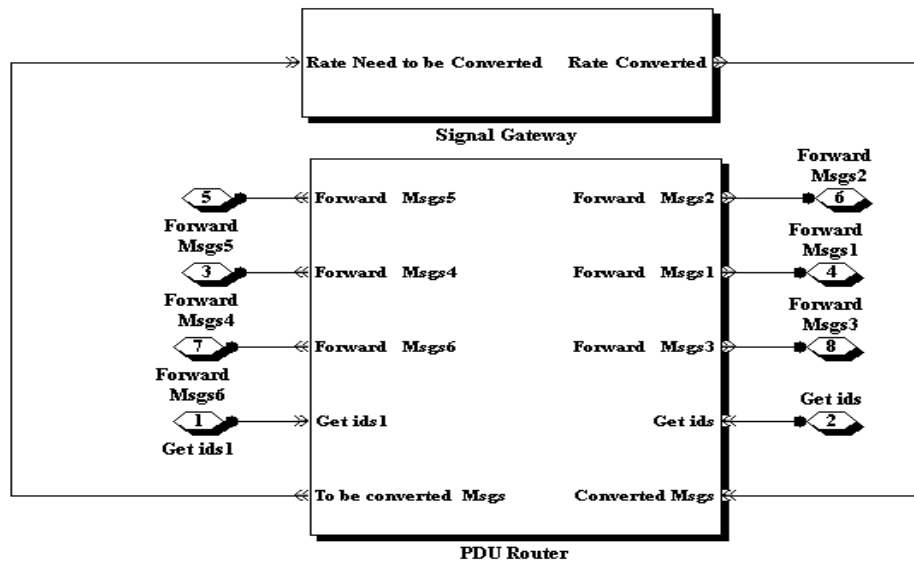


Figure 9.11: Gateway Simulation Model

9.3.1 COM MESSAGE GENERATION

The COM message generation includes setting up each message attributes, which are message length, message data, message priority number and message rate.

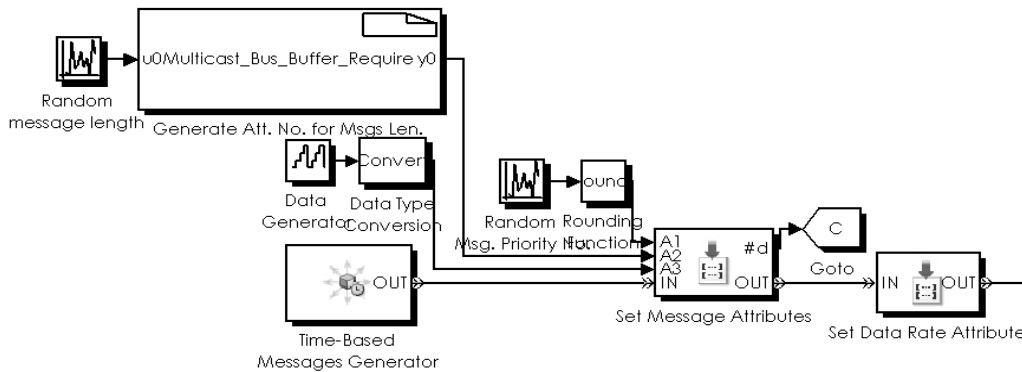


Figure 9.12: Random Message Generation

All these attributes should be generated randomly, so that each message with these attributes can be regarded as real world. The Uniform Random Number block can generate these attributes. Table 9.6 shows the details of these attributes.

Attribute Names	Attribute Value
Message Length	0 ~ 4095 Bytes
Message Data	8 Bits
Message Priority Number	1 ~ 200
Message Rate	125, 250, 500, 1000 Kbits/Sec

Table 9.6: Attribute Details

In these attributes, the “Message Length” is a special one. In the later design stages, the buffer selection is directly decided by its value. Therefore, the S-function Builder of Simulink is used. Here the development of the S-function using C is described. The inputs/outputs of this S-function are shown in Table 9.7. In this table, the input is the length of each message. Note that a variable length message is limited to a maximum of 4,095 bytes by the COM specification. To make the outputs of this S-Function simple, it was decided to use representative values to indicate each message length range.

S-Function: Multicast_Bus_Buffer_Require	
Input (Message Length)	Output (Representative Value)
(0, 1500) Bytes	1
(1500, 3000) Bytes	2
(3000, 4095) Bytes	3

Table 9.7: S-Function Input/Output

The S-Function Builder implements simple functions such as converting the input to get the output. Therefore, it was decided to use this block so that the communication with the C prototype can be developed.

Figure 9.13 shows the S-Function Builder, in which the C prototype was integrated into the Simulink environment.

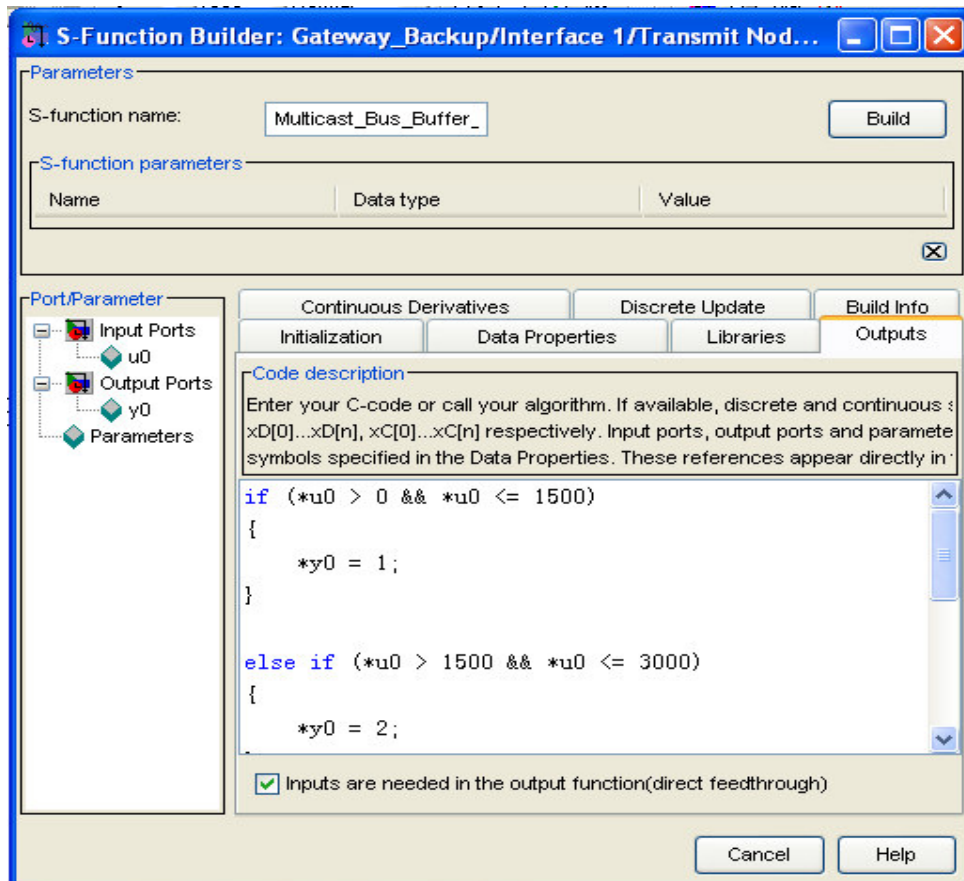


Figure 9.13: Message Length S-Function

Each ***S-Function Builder*** of the system generates its own S-Function wrapper C-file. This C file contains all the methods that Simulink calls when a simulation is running. For example, each message is Packed/Unpacked in the simulation. In this case the new S-Function Builder Generate Attr. No. for Msgs Len. Block is called. The S-Function then calls its “Multicast_Bus_Buffer_Require_wrapper.c” class, which contains functions called when initializing or calculating outputs. When the simulation needs to calculate the Multicast Bus Buffer Require blocks outputs it goes through the following process shown in Figure 9.14.

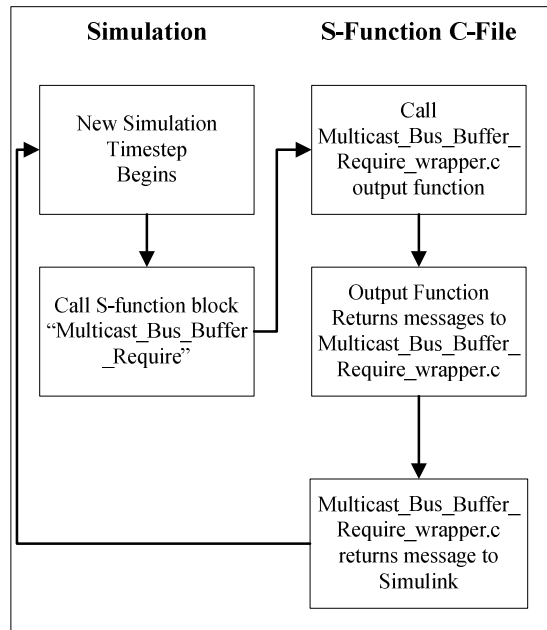


Figure 9.14: S-Function Process

9.3.2 BUFFER SELECTION

As discussed in the previous section, each message is randomly assigned a message length attribute. With this attribute, when each message goes into the buffer, it will select a buffer with an equivalent size. For example, if a message length is between 1500 and 3000 bytes, it will select a buffer size equal to 3000 bytes. However, if there are no buffers available, messages will be dropped into the Msg. Lost block. Figure 9.15 shows the simulation model of this buffer selection mechanism.

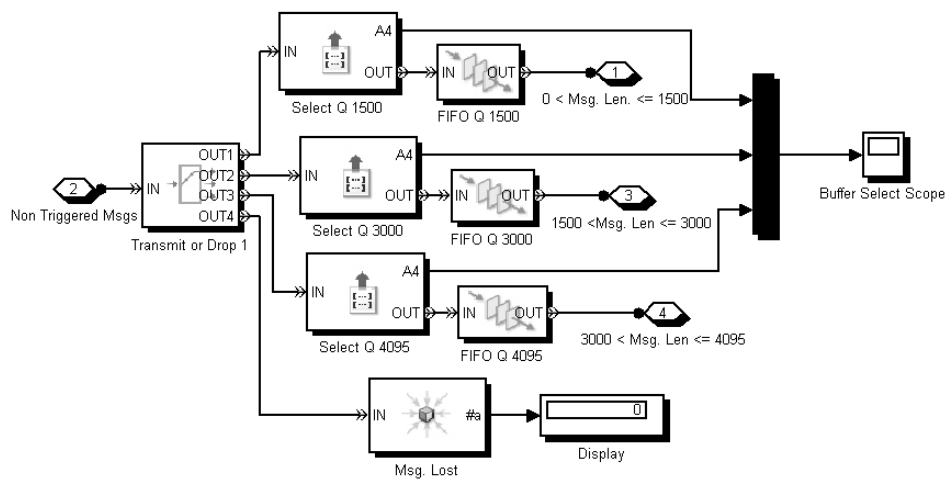


Figure 9.15: Buffer Selection

9.3.3 MESSAGE ROUTING

The PDU Router in the gateway is very complex; therefore, lots of functions of Simulink are used.

The main task of the PDU router is to control the network traffic, so it must have a routing table, which contains all the information on network events. For example, the routing table decides if the transmission is TP or Non-TP according to each message Interface ID (Table 9.9). The PDU router module is defined to simulate according to this information. The lookup table block can work as a routing table, statically storing all the messages' information like Interface ID, Message Priority Number and Device Number. Table 9.9 shows an example of a Routing Table while its explanations are described in Table 9.8.

Value Name	Description
Msg. Name	Indicates message function
Msg. Priority	Used to determine messages queuing sequence
Interface ID	Indicates which network each message should go to

Table 9.8: Routing table parameter explanations

Msg. Name	Device No.	Msg. Priority	Interface ID
DLCT (Drive Line Controller)	1	13	2
TCU (Transmission Control Unit)	2	35	2
EMS (Engine Management System)	3	67	1
SCS (Slip Control System)	4	45	2
IPK (Instrument Pack)	5	34	2
SAS (Steering Angle Sensor)	6	17	3
AIR_SUS (Air Suspension Controller)	7	110	3
ARC (Automatic Roll Control ECU)	8	60	1
TLM (Telemetric Unit)	9	43	1
HEVAC (Heating Ventilation and Air Conditioning Unit)	10	55	1
EPB (Electric Parking Brake)	11	65	3
ACC (Automatic Cruise Control)	12	70	1
VCS (Vehicle Control System)	13	120	3

Table 9.9: Example of a routing table

The Direct Lookup Table block is used to implement this routing table. The direct lookup table block uses its block inputs as zero-based indices into an n-d table. Users define a set of output values as the Table data parameter. Additionally, specify users what objects the input selects from the table: an element, a column, or a 2-D matrix.

In the example, shown in Figure 9.16, the Direct Lookup Table block has two inputs, both from the “msg_ids” of Get Attribute block. Figure 9.16 illustrates how it works.

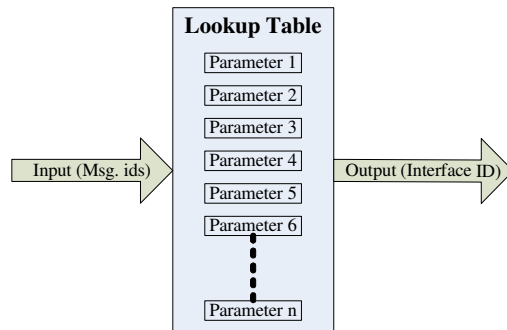


Figure 9.16: Routing table steps

Figure 9.17 shows that the direct lookup table block is succeeded by an S-Function Builder block. The S-Function builder converts the inputs to their corresponding output values, and then gives the Output Msgs to Diff. Interfaces block the switch sequence.

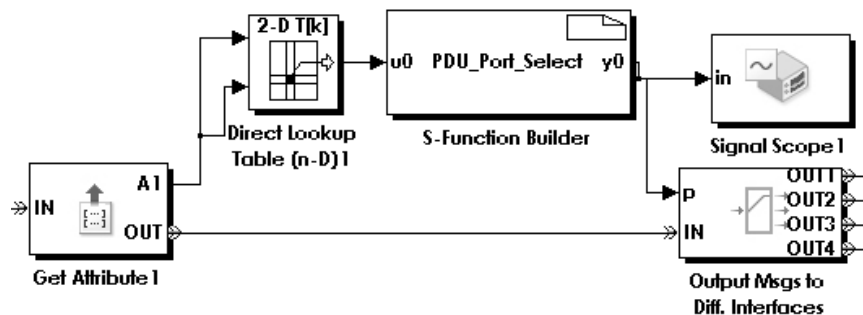


Figure 9.17: Routing Table Block

9.3.4 RATE CONVERSION

S-Function Builder blocks are used to implement the data rate setup. If the bus rate of the incoming message is the same as the bus rate of the interface it goes to, the S-Function builder block passes it directly to the corresponding interface. Otherwise, the S-Function Builder will forward this message to the Upper layer, so that the signal gateway can reset the data rate of this message to match its corresponding interface. An example of a Rate Convert_125 block is shown in Figure 9.18, whereby its input is the attribute data rate from the Get Attribute block, the Output Switch decides which way the message should go based on the output of the S-Function Builder, that

is, either go directly to the corresponding node or to the signal gateway of the upper layer.

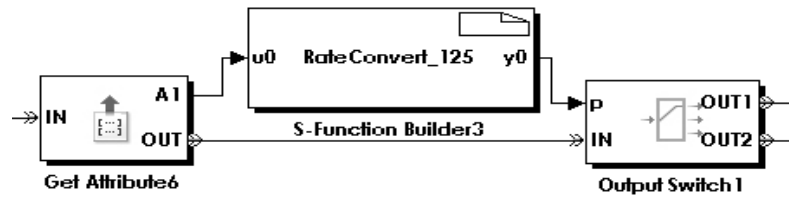


Figure 9.18: S-Function Builder for Rate Convert

9.4 TRANSMISSION TYPE MODELLING

Since the messages can be transmitted as either time-triggered or event-triggered, it is necessary to implement these features in the simulation. In the AUTOSAR defined Gateway specification, time-triggered transmission assumes the name Non-triggered TX, while the event-triggered transmission assumes the name Triggered TX.

- **Non-triggered Transmission (TP or Non-TP)**

In SimEvents, a block called Time-based Event Generator is the ideal option for Non-triggered message generation. In this case, the block generates each message based on a constant time.

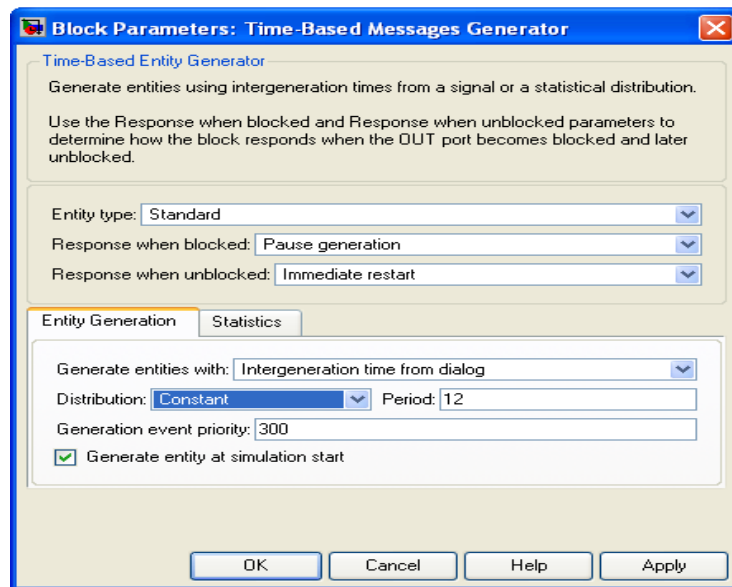


Figure 9.19: Non Triggered Message Generation

Figure 9.20 shows the Non-TP TX and TP TX without triggered blocks. As described in the Message Routing section, the routing table decides if a transmit type is TP or Non-TP. If a message has the same destination and original Interface ID, it is decided that this transmit type is Non TP. If a message has different destination and original Interface IDs, this transmit type is regarded as TP.

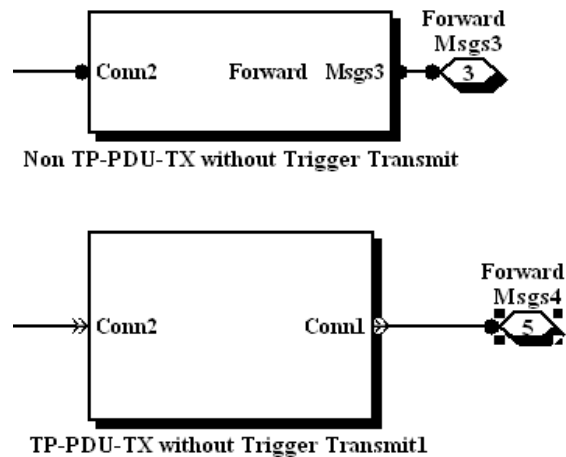


Figure 9.20: Non-triggered Transmission (TP or Non-TP)

- **Triggered Transmission (TP or Non-TP)**

For the Triggered TX, some special blocks have to be used. After doing research on the Simulink and SimEvents package, blocks like Goto, From, Enabled Gate can be implemented together to solve the problem.

The Goto block passes its input to its corresponding From blocks. Therefore, these two blocks can be regarded as a commanding side and a commanded side. The Enabled Gate block is usually connected with a From block. Once the From block is triggered by a signal from a Goto block, the Enabled Gate block will open its gate to let the messages pass through. Since the triggered function is also a part of the gateway, the detailed figure on this will be shown in the Gateway ECU simulation section.

Figure 9.21 shows one part of a triggered transmission in a PDU router. In this figure, the Goto block B sends a transmit request to receive sides with a relevant “msg_id”. The PDU router will not forward message until it receives a triggered command from a receive side. The Enable Gate block will then enable its gate once it gets a signal from the From block A, which is from the receive side.

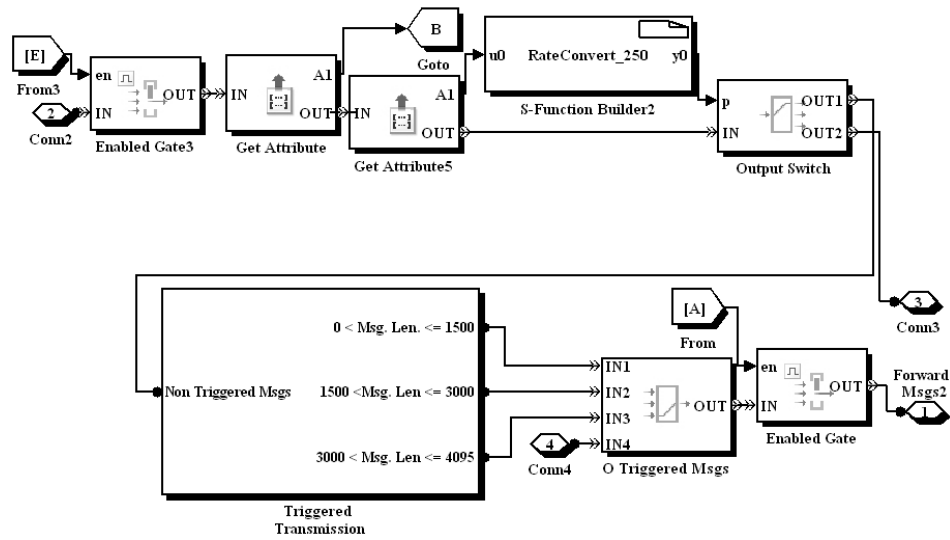


Figure 9.21: Triggered Transmission in Gateway Simulation Model

Figure 9.22 shows another part of a triggered transmission in receive side. In this figure, the receive side gets a receive indication signal from From block B. It will not, however, receive the message straight away. Instead it waits for the Single Server block to send a triggered command to the Goto block A, which is related to the PDU router.

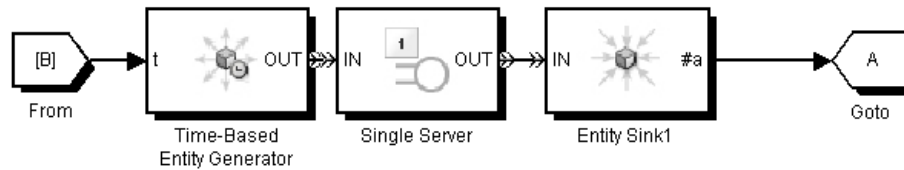


Figure 9.22: Triggered Transmission in Receive Node Simulation Model

The Non-TP TX and TP TX mechanism in this Triggered Transmission is the same as in the Non Triggered Transmission.

9.5 RECEPTION TYPE MODELLING

A receive node is very straightforward; it should be configured with a filter to get rid of unwanted messages. If this receive node wants to receive message periodically, it should use Time-Based Entity Generator, an Enabled Gate, a From and Goto blocks. Figure 9.23 shows the periodic reception of a receive node. This procedure is described below:

- Time-Based Receive Indication Generator block generates an indication periodically in a certain time.
- The Goto 2 block is triggered by the Time-Based Receive Indication Generator block every so often. It will then send a signal to the From 1 block.
- Once the From 1 block gets a signal from the Goto 2 block, it will enable the Enabled Gate block, allowing it to receive the messages periodically.

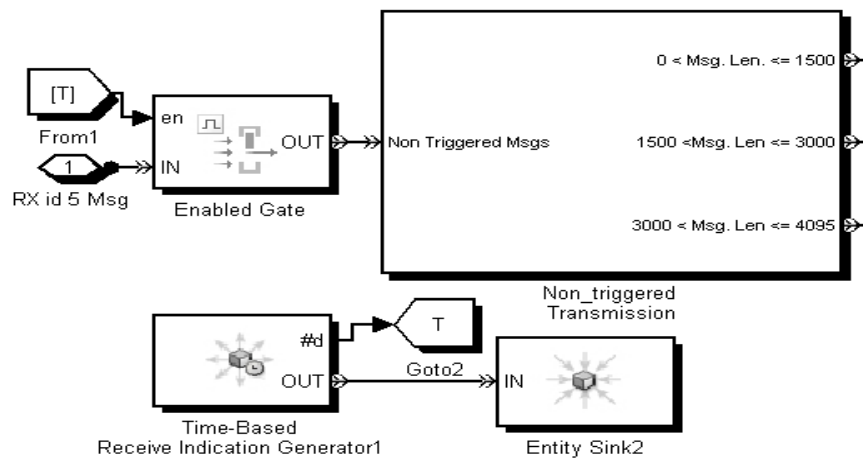


Figure 9.23: Time Scheduled Messages Reception

As defined in the AUTOSAR Gateway Specification, if the Reception type is Non-TP, an incoming message must be from the same originating Interface. If the Reception type is TP, an incoming message must be from a different originating Interface.

9.6 SIMULATION MODEL DESIGN VALIDATION COMMENTS – FROM AN EXPERT IN AUTOMOTIVE INDUSTRY

This in-vehicle network simulation model was sent to an expert in ATUOSAR to evaluate, below is this expert's general comments:

“One of the things that don't seem to be covered is the fact that TP routing on the fly isn't treated in detail. For IF routes I recall that buffering was a property of a route, whereas for TP routes the idea seems to be to define a pool of memory that got allocated to TP request dynamically. While this might be useful to cover, it is probably sensible to skirt around the issue. As implementers of the PDUR for AUTOSAR's validator, no-one ever managed to explain to us what the spec meant here and how it was supposed to work without dynamic memory allocation (which you typically cannot do on an embedded target)... I fear the requirements here were a bit like "It has to have a 45" screen and still fit into a purse or wallet.”

“However, the overall impression is good – it is an interesting experiment!!”

As a result of these comments the following changes were made to the model:

- TP routing is mentioned in Section 9.3.3 and 9.4.
- Buffer lengths are setup statically rather than dynamically, mentioned in Section 9.3.2.

9.7 REFERENCES

AUTOSAR (2006) Specification of PDU Router, AUTOSAR GbR.

INFINEON Infineon C116 MCU. Infineon.

MATHWORKS, T. (1997a) Matlab and Simulink User's Guides. The MathWorks, Inc.

MATHWORKS, T. (2006) SimEvents User's Guides, The MathWorks, Inc.

PEAK PCAN Explore 3. PEAK System.

PART FOUR

TESTING AND CONCLUSION

10.1 INTRODUCTION

“One of the most important and difficult tasks facing a model developer is the verification and validation of the simulation model.” (BANKS, 2000A) Conceptually, the verification and validation process consists of the following:

- Verification is concerned with building the model correctly. Two questions are asked:
 1. Is the model implemented correctly?
 2. Are the input parameters and logical structure of the modelled system correctly represented?
- Validation is concerned with building the right model, according to the system requirement.

This process is used to determine whether a simulation model is an accurate representation of the real system. The basic idea of this process is to repeatedly compare the created model to the actual system behaviour, while using the discrepancies between the two to develop an accurate model.

This chapter focuses on the testing process by using the verification and validation processes. Figure 10.1 shows the stages of Model Building, Verification and Validation. (BANKS, 2000C)

1. The first step is to analyse the real system to be simulated and the interactions among its various components and collect data on its behaviour.
2. The second step is to construct a conceptual model with the collection of the components and data from the first step, as well as with the hypotheses on the values of model input parameters.

3. The third step is the translation of the operational model into a computer-recognizable form. Model building, however, is not a linear process with these three fixed steps. The simulation model has to be iterated through each of these steps lots of times while building, verification, and validation.

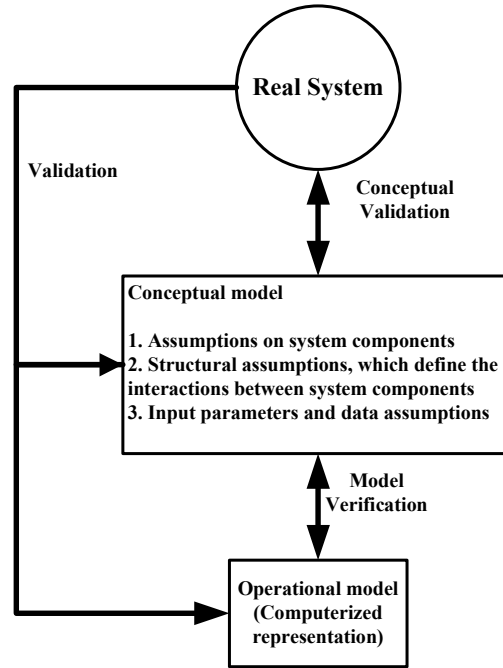


Figure 10.1: Model Testing

10.2 TESTING TOOLS

There are three methods (MATHWORKS, 1997A) available in Simulink for analysing the outputs from the simulation model, these methods are:

- Feed a signal into a Scope block (Real-time Graphic Representation).
- Use Matlab plotting commands to get the return variables of the output.
- Using Matlab plotting commands and To Workspace block to output results to the workspace.

During the gateway simulation model design, the most common testing methods are Real-time Graphic Representation (Scope) and Matlab Workspace.

10.2.1 REAL-TIME GRAPHIC REPRESENTATION

The Simulink and SimEvents packages of Matlab provide the Scope and Signal Scope blocks for analysing the results of simulation models.

The Scope and Signal Scope blocks display signals generated during a simulation. The advantages of these blocks for designing an in-vehicle network gateway are to adjust the amount of time and the range of messages displayed. The Scope window and Scope's parameter values can be moved, resized and modified during the simulation.

Figure 10.2 shows an example of the routing information Scope in the PDU Router. In this Scope, the Y axis displays the message's ids, and the X axis displays the timing, showing how often the routing table in the PDU Router checks each message.

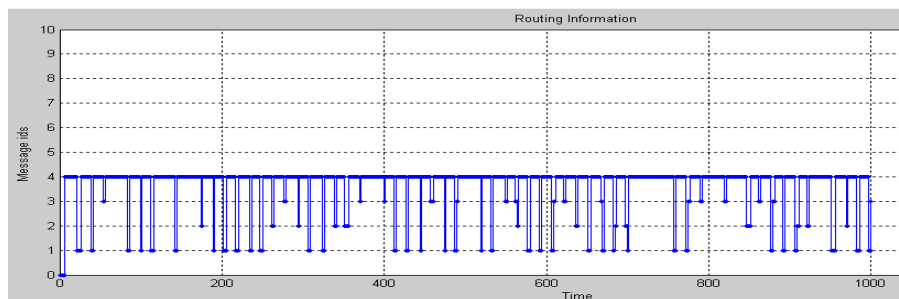


Figure 10.2: Routing Information Scope

Figure 10.3 shows an example of the Buffer Selection Scope in one of the receive nodes. In this Scope, the Y axis displays the Message Length Representation Value, which is described in the Buffer Selection section in the Gateway Specification chapter. The X axis displays the timing, showing how often the selected buffer stores each message.

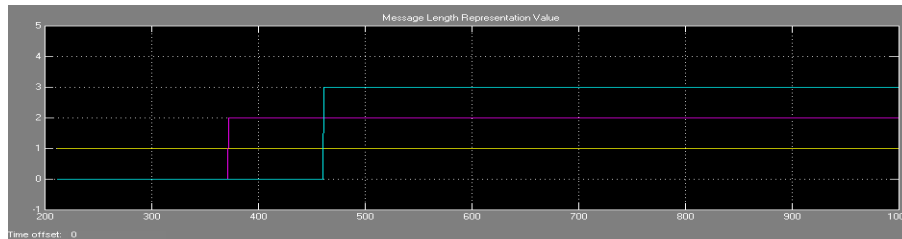


Figure 10.3: Buffer Selection Scope

10.2.2 MATLAB WORKSPACE

The Matlab Workspace can work as either inputs or outputs for the simulation model testing. As mentioned in the Gateway Specification chapter, a Local Database of the routing table (Loaded from excel files) is stored in the Workspace, as shown in Figure 10.4.

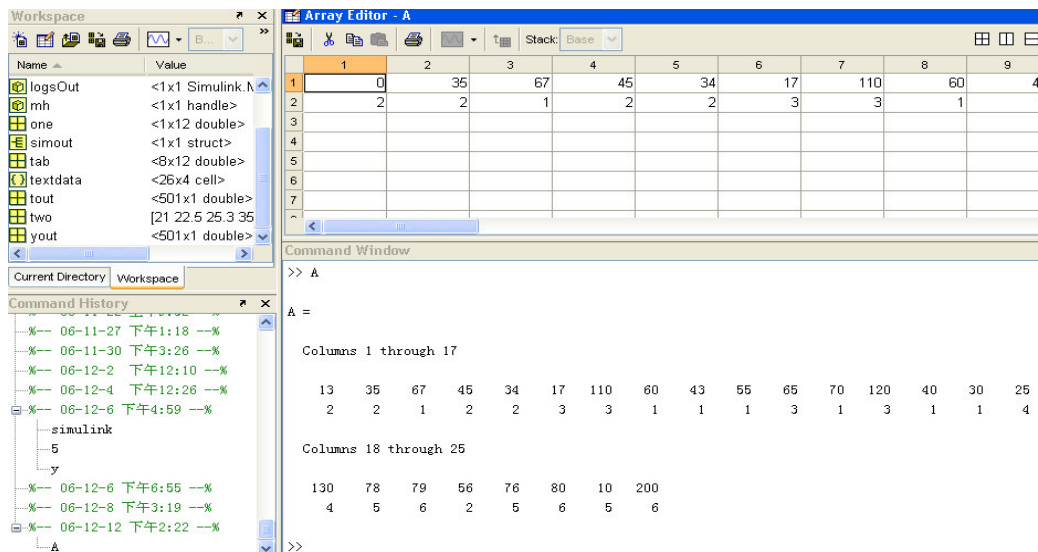


Figure 10.4: Routing Table Workspace

For output testing, simply pausing the simulation at any time and using the main Matlab window to call up the Local Database can access the local database. The Local Database can also be passed into the SimEvents model and displayed on the Display Blocks.

10.3 VERIFICATION OF GATEWAY SIMULATION MODEL

The aim of model verification is to ensure that the conceptual model is accurately reflected the computerized representation. The conceptual model involves some abstraction on system operations, or some amount of simplification of actual operations.

In the next subsection, the Verification Testing Case Table is defined. In this table, there are the following components, described in Table 10.1:

Component Name	Description
Case	Indicates the test case name
Subsystem/Component (Blocks)	Indicates the subsystems and blocks simulated in the Matlab & Simulink
Date	Indicates the date that each test case is executed.
Version	Indicates the test tool version
Test Case Description	Explains each test case function
Test Procedure	Explains how each case is tested
Expected Results	Explain the results of a successful test
Failure	Explain the results of a failed test
Result	Indicates each test case result

Table 10.1: Verification Testing Case Table Components Description

10.4 VERIFICATION TESTING CASES

Message Generation:

Test Case 1:

Case	Subsystem / Component (Blocks)	Date	Version
Event-triggered Messages Generation	ECU 1, Transmit Node 1: Event-base Message Generator (message length, data, priority number, data rate) Event-triggered Messages Checking Scope	02-12-06	MATLAB R2006a
Test Case Description: It generates messages with attributes of length, data rate, and priority number.			
Test Procedure: Connecting four <u>Scope</u> blocks to the signal outputs of the <u>Set Message Attributes</u> block and observing results from those four Scope blocks.			
Expected Results: if each generated message is attached four different attributes (length, data, data rate, and priority number) correctly.			
Failure: if each generated message is not attached any of its attributes or its attached attributes has wrong format.			
Result: Pass			

Test Case 2:

Case	Subsystem / Component (Blocks)	Date	Version
Time-triggered Messages Generation	ECU 3, Transmit Node 3: Event-base Message Generator (message length, data, priority number, data rate) Time scheduled Table Time-triggered Messages Checking Scope	02-12-06	MATLAB R2006a
Test Case Description: It generates messages with attributes of length, data rate, and priority number.			
Test Procedure: Connecting four <u>Scope</u> blocks to the signal outputs of the <u>Set Message Attributes</u> block, getting results from those four Scope blocks, and comparing the scope time to the time in Time scheduled Table. .			
Expected Results: if each message is generated according to the Time Scheduled Table and generated message is attached four different attributes (length, data, data rate, and priority number) correctly.			
Failure: if each message is not generated according to the time of the schedule table and each generated message is not attached any of its attributes or its attached attributes has wrong format.			
Result: Pass			

Transmission:

Test Case 3:

Case	Subsystem / Component (Blocks)	Date	Version
Non TP-PDU-TX without Trigger Transmit	Gateway ECU, PDU router, Non TP-PDU-TX without Trigger TX: Enabled Gate Trigger Transmit Checking Scope	02-12-06	MATLAB R2006a
Test Case Description: Messages are transmitted right after a transmit request without any delay.			
Test Procedure: Monitoring the <u>Trigger Transmit Checking Scope</u> to see if the message is right after a transmit request.			
Expected Results: if the message to be transmitted is right after a transmit request without any delay.			
Failure: if there is no transmit request before a message to be transmitted, or there is no message following a transmit request.			
Result: Pass			

Test Case 4

Case	Subsystem / Component (Blocks)	Date	Version
Non TP-PDU-TX with Trigger Transmission	Gateway ECU, PDU router, Non TP-PDU-TX without Trigger TX: ECU 1, Receive Node 1: Enabled Gate Time Based Event Generator Trigger Transmit Checking Scope	02-12-06	MATLAB R2006a
Test Case Description: Messages are transmitted after a trigger command from a reception side.			
Test Procedure: Monitoring the <u>Trigger Transmit Checking Scope</u> to see if a message is triggered by a trigger command, which means this transmission allows a certain time delay.			
Expected Results: if after a transmit request, there is no message following, and the expected message is transmitted by a trigger command from a receive side.			
Failure: if the message is still transmitted right after the transmit request, or there is no message to be transmitted after a trigger command from the receive side.			
Result: Pass			

Test Case 5

Case	Subsystem / Component (Blocks)	Date	Version
TP-PDU-TX	Gateway, PDU Router: Enabled Gate Transport protocol 1 Scope Transport protocol 2 Scope	02-12-06	MATLAB R2006a
Test Case Description: Data transmission happens via the transport protocol modules. The transmit request is forwarded by the gateway ECU to the related TP module.			
Test Procedure: Checking Transport protocol 2 Scope, if there is a message coming from Transport protocol 1. Checking Transport protocol 1 Scope, if there is a message coming from Transport protocol 2.			
Expected Results: if message transmission is between the same or different transport protocols.			
Failure Recovery: if message transmission is not between different transport protocols but between different interfaces.			
Result: Pass			

Reception:

Test Case 6:

Case	Subsystem / Component (Blocks)	Date	Version
Non-TP-PDU RX	ECU 1, Receive Node 3: Enabled Gate Protocol Interface 1 Scope Protocol Interface 2 Scope	02-12-06	MATLAB R2006a
Test Case Description: The reception of the messages happens between different protocol interfaces.			
Test Procedure: Checking Protocol Interface 2 Scope, if there is a message coming from Transport protocol interface 1. Checking Protocol Interface 1 Scope, if there is a message coming from protocol interface 2.			
Expected Results: if message reception does happen between different interfaces.			
Failure Recovery: if message reception does not happen between different interfaces.			
Result: Pass			

Test Case 7:

Case	Subsystem / Component (Blocks)	Date	Version
TP-PDU RX	Gateway, PDU Router: Enable Gate Transport protocol 1 Scope Transport protocol 2 Scope	02-12-06	MATLAB R2006a
Test Case Description: Data to be received is from a transport protocol module.			
Test Procedure: Checking Transport protocol 2 Scope, if there is a message coming from Transport protocol 1. Checking Transport protocol 1 Scope, if there is a message coming from Transport protocol 2.			
Expected Results: if message transmission does happen between different transport protocols.			
Failure Recovery: if message transmission does not happen between different transport protocols.			
Result: Pass			

Gateway:

Test Case 8:

Case	Subsystem / Component (Blocks)	Date	Version
Non TP-PDU-Gateway without rate conversion (Non-trigger)	Gateway, PDU Router, Non TP-PDU-TX without Trigger Transmit: Protocol Interface 1 Scope Protocol Interface 2 Scope Data Rate Checking 1 Scope Data Rate Checking 2 Scope	02-12-06	MATLAB R2006a
Test Case Description: It acts as direct gateway between two interface modules. Messages received from one interface shall be forwarded to the other interface. The transmission of the gateway works the same as the Non-trigger TX.			
Test Procedure: Checking data rate signal has any difference from Data Rate Checking 1 Scope to Data Rate Checking 2 Scope. The Non-trigger test procedure is the same test case 3.			
Expected Results: if messages transferred from one network to another network and both networks with the same data rate do not need rate conversion.			
Failure: if messages transferred from one network to another network and both networks with the same data rate do need rate conversion.			
Result: Pass			

Test Case 9:

Case	Subsystem / Component (Blocks)	Date	Version
Non TP-PDU-Gateway without Rate Conversion (Trigger)	Gateway, PDU Router: Protocol Interface 1 Scope Protocol Interface 2 Scope Data Rate Checking 1 Scope Data Rate Checking 2 Scope	02-12-06	MATLAB R2006a
Test Case Description: It acts as direct gateway between two interface modules. Messages received from one interface shall be forwarded to the other interface. The transmission of the gateway works as the same as the trigger TX.			
Test Procedure: Checking data rate signal has any difference from Data Rate Checking 1 Scope to Data Rate Checking 2 Scope. The Non-trigger test procedure is the same test case 4.			
Expected Results: if messages with the same data rate do transmission and reception between different interfaces. Also the message transmission works the same as the triggered transmission.			
Failure: if messages with the same data rate do not transmit and receive between different interfaces. Also the message transmission does not work the same as the triggered transmission.			
Result: Pass			

Test Case 10:

Case	Subsystem / Component (Blocks)	Date	Version
Non TP-PDU-Gateway with Rate Conversion (Trigger)	Gateway, PDU Router, Signal Gateway, Rate Conversion: Input Switch, Output Switch, New Rate Setup, Rate Conversion Display.	02-12-06	MATLAB R2006a
Test Case Description: It is not a direct gateway between two interface modules. Messages received from one interface forwarded to a different rate interface needs to be converted. The transmission of the gateway works as the same as the Non-trigger TX.			
Test Procedure: Checking data rate signal has any difference from Data Rate Checking 1 Scope to Data Rate Checking 2 Scope. The Non-trigger test procedure is the same test case 4.			
Expected Results: if messages with different data rate do transmission and reception between different interfaces. Also the message transmission works the same as the triggered transmission.			
Failure: if messages with different data rate do not transmit and receive between different interfaces. Also the message transmission does not work the same as the triggered transmission.			
Result: Pass			

Test Case 11:

Case	Subsystem / Component (Blocks)	Date	Version
TP-PDU Gateway	Gateway, PDU Router, TP-PDU-TX with Triggered, TP-PDU-TX without Triggered: TP-PDU Gateway Input Switch TP-PDU Gateway Output Switch Transport protocol 1 Scope Transport protocol 2 Scope	02-12-06	MATLAB R2006a
Test Case Description: Messages transmission and reception happen between two different transport protocols.			
Test Procedure: Checking the Transport protocol 2 Scope, if there is message transmission and reception in the Transport protocol 1 Scope. Or checking the Transport protocol 1 Scope, if there is message transmission and reception in the Transport protocol 2 Scope.			
Expected Results: if messages transfer happen within the transport protocols of gateway.			
Failure Recovery: if messages transmission and reception do not happen within the transport protocols of gateway.			
Result: Pass			

Communication Bus

Test Case 12:

Case	Subsystem / Component (Blocks)	Date	Version
Bus Loading	Bus: Gain ,Divide Average Arrival Rate Slip Bar Bus Loading Checking Scope	02-12-06	MATLAB R2006a
Test Case Description: This case tests if the Bus Loading changes according to the Average Arrival Rate Slip Bar movement.			
Test Procedure: Monitoring the <u>Bus Loading Checking Scope</u> change by moving the <u>Average Arrival Rate Slip Bar</u> .			
Expected Results: if the bus loading is calculated by the maths function blocks Gain and Divide, and once moving the Average Arrival Rate Slip Bar, the value of the bus loading will be changed.			
Failure Recovery: if the bus loading is calculated by the maths function blocks Gain and Divide, and once moving the Average Arrival Rate Slip Bar, the value of the bus loading does not change.			
Result: Pass			

Buffer Selection**Test Case 13:**

Case	Subsystem / Component (Blocks)	Date	Version
Buffer Selection	ECU3, Buffer Selection: FIFO Queues Buffer Selection S-function Builder, Buffer Selection Output Switches, Message Length Representation Value Scope	02-12-06	MATLAB R2006a
Test Case Description: This case tests if an incoming message is selecting the right size of the buffer.			
Test Procedure:			
Expected Results: if a message selects a buffer whose size is equivalent to its message length.			
Failure: if a message does not select a proper buffer or selects a wrong buffer.			
Result: Pass			

Message Routing**Test Case 14:**

Case	Subsystem / Component (Blocks)	Date	Version
Routing Table	Gateway ECU, PDU Router: Direct Lookup Table PDU_Port_Select (S-Function Builder) Local Database (Matlab Workspace) Routing Table Information Scope	02-12-06	MATLAB R2006a
Test Case Description: The lookup table block can work as a routing table statically storing all the messages information like Network Number, Message Priority Number, and Device Number.			
Test Procedure: Comparing the routing information in the Routing Table Information Scope to the values in the Local Database.			
Expected Results: if a coming message is sent to its corresponding network according to the information in the routing table of gateway.			
Failure Recovery: if a coming message is not sent to its corresponding network according to the information in the routing table of gateway.			
Result: Pass			

Rate Conversion**Test Case 15:**

Case	Subsystem / Component (Blocks)	Date	Version
Message Rate Conversion	Gateway ECU, PDU Router, Rate Conversion: Rate_Convert (S-Function Builder)	02-12-06	MATLAB R2006a
Test Case Description: This case tests if an incoming message with different data rate is to be converted to the right data rate corresponding to the network it goes to.			
Test Procedure: Rate Conversion works the same as Case 10.			
Expected Results: if a message goes to a different data interface or network, and this message's data rate is converted.			
Failure: if a message goes to a different data interface or network, this message's data rate is not converted.			
Result: Pass			

10.5 VALIDATION OF GATEWAY SIMULATION MODEL

10.5.1 OBJECT OF VALIDATION

“Validation is the overall process of comparing the model and its behavior to the real system and its behavior.” (BANKS, 2000C)

To optimise the gateway performance, it is necessary to evaluate the gateway model by changing each model parameter against some criteria in the real system. At this stage of this research, these criteria include:

1. Number of dropped messages
2. Throughput rate
3. Average message latency for an individual message
4. Average message length
5. Buffer utilization

10.5.2 VALIDATION PROCEDURE

Naykir abd Finger [1967] formulated a three-step approach (BANKS, 2000C) as an aid to the validation process, which has been widely followed:

1. Build a model that has high face validity.

This step involves the contraction of a model that is reasonable on its face, to model users and others who know the real system being simulated.

Sensitivity analysis can be applied to measure a simulation model's face validity. The model is questioned if it behaves unexpectedly when input variables are changed. For example, in vehicle network, if the arrival rate of messages were to increase, the utilization of the servers' message waiting time would also be increased. Therefore, in this case the model builder would have to define some ideas for the output model when the input variables are either increased or decreased.

2. Validate model assumptions.

There are two general classes of model assumptions: Structural assumptions and Data assumptions.

Structural Assumptions ask what the system operation is and deals with the simplification and abstractions of reality. For example, in vehicle network systems, messages may wait in one queue, or there may be an individual queue for transmit server or receive server. If there is more than one queue, messages could be transmitted or received following FIFO (first-in, first-out) mechanism, or messages could switch queues if one is executing faster. The number of transmit or receive servers may be fixed or variable. Therefore, these structural assumptions should be verified with transmit and receive servers according to the bus loading and real time implementation.

Data Assumptions should be based on the collection of reliable data and correct statistical analysis of the data. For example, in vehicle network simulation models, data is collected from:

- Inter-arrival times of messages during specific periods of peak bus loading (Heavy Traffic).
- Transmit or receive time for each single message from one interface to another interface in different bus loading.
- Transmit or receive time for each single message from one network to another network in different bus loading.
- Buffer selection and implementation time for messages with different lengths.

3. Compare the model input-output transformations to corresponding input-output transformations for the real system.

The ultimate and only objective test of a simulation model is the ability to predict the future performance of the real system by simulating the real systems' inputs and policy. Additionally, in the real system, when some input variables increase or decrease, the model should accurately predict what is happening and simulate those circumstances.

In this validation step, the simulation model is observed as an input-output transformation. Essentially, that the simulation model accepts the input parameters and transforms these inputs into output measures of performance.

The gateway simulation model validation follows these three steps. At the start of this validation process, the overall simulation model was sent to an expert, who works in AUTOSAR and is familiar with all aspects of AUTOSAR's Gateway Specification. This expert's comments are in the Section 9.6 of Chapter Nine. Figure 10.5 shows the basic structure, which can represent the procedure of gateway validation.

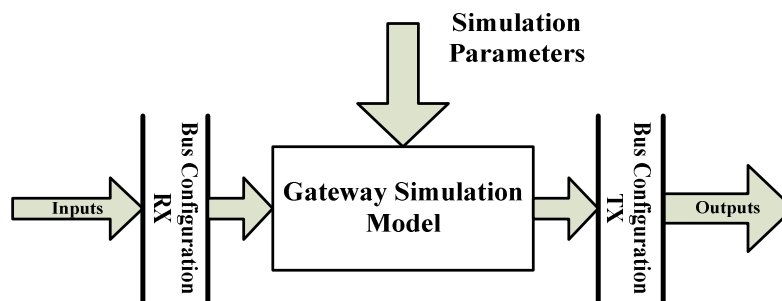


Figure 10.5: Gateway Validation Procedures

In Figure 10.5, on both sides of the Gateway Simulation Model, there are two bus configurations: Bus Configuration RX and Bus Configuration TX. Different block components and bus parameters are defined during the testing.

In Bus Configuration RX block, there are two components, namely Bus Controller and Bus Interface. Table 10.2 describes this Bus Configuration RX block.

Block Component	Description
Bus Controller	This component has one parameter: <i>H/W Buffer Length</i> : it sets up the buffer length in the controller
Bus Interface	This component has one parameter: <i>RX Overhead Time (ms)</i> : it indicates how long a bus interface takes to receive a message.

Table 10.2: Bus Configuration RX

The Gateway Simulation Model contains a PDU Router and Signal Gateway. Each has different block components and gateway parameters. Table 10.3 describes this Gateway Simulation Model.

Block Component		Description
PDU Router	Router	This component has two parameters: <i>Cycle Time (ms)</i> : it indicates how often a PDU Router opens its gate to process an amount of messages. <i>Table Size</i> : it indicates the size of a lookup table in the PDU router.
	Transmit	This component also has one parameters: <i>H/W Buffer Length</i> : it indicates the size of a transmit buffer in the PDU Router for transmitting the messages onto the bus. <i>Buffer Type</i> : it indicates the queuing mechanism for those messages temporarily stored in the RAM. (<i>FIFO or Priority</i>)
Signal Gateway		This component has only one parameter: <i>Rate Conversion</i> : it indicates the time needed for a message's rate to be converted.

Table 10.3: Gateway Simulation Model

In a Bus Configuration TX block, there are two components, namely a Bus Controller and a Bus Interface. Table 10.4 describes this Bus Configuration TX block.

Block Component	Description
Bus Interface	This component has one parameter: <i>Buffer Length</i> : it indicates the size of a RAM, which can store the coming messages temporarily, when the buffer in the controller is full. <i>TX Overhead time (ms)</i> : it indicates how long a bus interface takes to forward a message to the bus controller.
Bus Controller	This component has two parameter: <i>H/W Buffer Length</i> : it indicates the size of a transmit buffer in the bus controller for transmitting the messages onto the bus. <i>TX Time (ms)</i> : it indicates how long a bus controller takes to transmit a single message onto the bus according to the bus loading and bus rate.

Table 10.4: Bus Configuration TX

In the testing, some useful data will be collected after each simulation run. Table 10.5 defines the gateway evaluation case as well as describing the results.

Component	Description
Average Queue Length	It indicates the length in the buffer or RAM
Average Message Delay	It indicates the average time each message stages in the buffer.
Number of Dropped Messages	It indicates the number of messages dropped during each step within each simulation run.
Throughput	It indicates the number of messages implemented during each step within each simulation run.
Buffer Utilization	It indicates the percentage of buffer used within each simulation run.
Gateway Performance Summary	It indicates the summary the results by summing the five results above.
Performance Improvement	It indicates the improvement by comparing current simulation result with the last simulation result.

Table 10.5: Gateway Model Evaluation Results

Based on the three-step approach and the validation process in Figure 10.5, four gateway validation steps are defined:

1. Select different Bus Configurations (Bus Loading and Bus Rate).
2. Select same sets of Simulation Parameters.
3. Run simulation for each combination of step 1 and 2.
4. Document and analyse the performance metrics from step 3.

10.6 OPTIMISATION OF GATEWAY PERFORMANCE

10.6.1 EVALUATION CASES TABLE

As mentioned in the Object of Validation section, five performance metrics need to be evaluated:

1. Number of dropped messages,
2. Throughput rate,
3. Average message latency for an individual message,
4. Average message length and
5. Buffer Utilization.

10.6.2 EVALUATION CASES

1. Same Bus Loading and Same Bus Rate

This section evaluates the gateway parameters by setting the bus loading and bus rate as the same value. Table 10.6 shows the initial setup in this case.

Parameter			Value	Remark
RX Bus	Bus Loading (%)		80	When the bus loading is 80% and the bus rate is 500Kbits/s , the msg/sec on the bus is 3389.8 . Suppose 20% of this msg/sec going to the gateway block is realistic, so every 1.48 ms, an msg is received by the RX bus.
	Bus Rate (Kbits/sec)		500	
TX Bus	Bus Loading (%)		80	When the bus loading is 80% and the bus rate is 500Kbits/s , the msg/sec on the bus is 3389.8 . Every 0.94 ms, an msg is transmitted by the TX bus.
	Bus Rate (Kbits/sec)		500	
RX	Bus Controller	Buffer Length (Msgs)	150	Set the RAM size is 1.2MB , each message has 8 Bytes. (Fixed)
	Bus Interface	RX Overhead Time(ms)	2	(Fixed)
PDU Router	Router	Cycle Time (ms)	2	The Router Cycle time could be 2 ms
		Table Size	25	
	Transmit	Buffer Length (Msgs)	170	Set the RAM size is 1.4MB , each message has 8 Bytes.
		Buffer Type	FIFO	The initial buffer type is set up as FIFO
Signal Gateway		Rate Conversion		
TX	Bus Interface	Buffer Length (Msgs)	150	Set the RAM size is 1.2MB , each message has 8 Bytes. (Fixed)
		TX Overhead time(ms)	5	The Overhead time could be 2~4 ms (Fixed)
	Bus Controller	H/W Buffer Length (Msgs)	10	(Fixed)
		TX Time (ms)	0.94	(Fixed)

Table 10.6: Initial Case Setup 1

Case 1:

Block		Parameter	Value	Avg. Queue Leng.	Avg. Msg Delay (ms)	No. of Dropped Msgs.	Through-put (Msg/sec)	Buffer Utilization (%)
RX	Bus Controller	H/W Buffer Length	150	86.3	28.56	40	500	57.5
	Bus Interface	RX Overhead Time(ms)	2					
PDU_R	Router	Cycle Time (ms)	2	111.6	249	80	249	65.6
		Table Size	25					
	Transmit	Buffer Length	170					
		Buffer Type	FIFO					
Signal Gateway		Rate Conver.			0			
TX	Bus Interface	Buffer Length	150	24.8	15.65	0	200	10.4
		TX Overhead time(ms)	5					
	Bus Controller	H/W Buffer Length	10	0	0	0	199	0
		TX Time (ms)	0.94					
Gateway Performance Summary					302.36	120	1148	
Performance Improvement (%)					0	0	0	0

In the first Case of this set, the *Buffer Type* in PDU Router is set up as the **FIFO**, which makes the gateway work very inefficiently, especially for the *Average Message Delay*.

Case 2:

Block		Parameter	Value	Avg. Queue Leng.	Avg. Msg Delay (ms)	No. of Dropped Msgs.	Through-put (Msg/sec)	Buffer Utilization (%)
RX	Bus Controller	H/W Buffer Length	150	86.3	28.56	40	500	57.5
	Bus Interface	RX Overhead Time(ms)	2					
PDU_R	Router	Cycle Time (ms)	2	111.6	72.18	80	249	65.6
		Table Size	25					
	Transmit	Buffer Length	170					
		Buffer Type	P					
Signal Gateway		Rate Conver.			0			
TX	Bus Interface	Buffer Length	150	53.74	24.8	0	200	35.8
		TX Overhead time(ms)	5					
	Bus Controller	H/W Buffer Length	10	0	0	0	199	0
		TX Time (ms)	0.94					
Gateway Performance Summary					125.54	120	1148	
Performance Improvement (%)					58.5	0	0	

When changing the Buffer Type as the **Priority**, the *Average Message Delay* is improved by **58.5** percent in Case 2, which means that each message is processed much quicker by using the **Priority** buffer. But there are no big differences for the other four results: *Average Queue Length*, *No. of Dropped Messages*, *Throughput* and *Bus Utilization*. In Case 2, however, the *Average Message Delay* in TX's Bus Interface is a little bit longer than in Case 1, this shows that, the TX Bus is subject to more pressure, if the gateway works too quickly.

Case 3:

Block		Parameter	Value	Avg. Queue Leng.	Avg. Msg Delay (ms)	No. of Dropped Msgs.	Throughput (Msg/sec)	Buffer Utilization (%)
RX	Bus Controller	H/W Buffer Length	150	86.3	16.15	40	500	57.5
	Bus Interface	RX Overhead Time(ms)	2					
PDU_R	Router	Cycle Time (ms)	1.5	83.91	26.53	0	332	49.4
		Table Size	25					
	Transmit	Buffer Length	170					
		Buffer Type	P					
Signal Gateway		Rate Conver.			0			
TX	Bus Interface	Buffer Length	150	61.58	36.19	33	199	41.1
		TX Overhead time(ms)	5					
	Bus Controller	H/W Buffer Length	10	0	0	0	198	0
		TX Time (ms)	0.94					
Gateway Performance Summary					78.87	73	1229	
Performance Improvement (%)					37.2	39.2	7	

In Case 3, the Cycle Time is shortened, which makes the *Average Message Delay* shorter compared to Case 2; the results are improved by **37.2** percent. The *No. of Dropped Messages* and *Throughput* are also improved by **39.2** and **7** percent. But, the *No. of Dropped Messages* in TX's Bus Interface is higher than in Case 1 and 2. Consequently, the TX Bus is under more pressure, if the gateway works too quickly.

Case 4:

Block		Parameter	Value	Avg. Queue Leng.	Avg. Msg Delay (ms)	No. of Dropped Msgs.	Throughput (Msg/sec)	Buffer Utilization (%)
RX	Bus Controller	H/W Buffer Length	150	86.3	28.56	40	500	57.5
	Bus Interface	RX Overhead Time(ms)	2					
PDU_R	Router	Cycle Time (ms)	2	122.5	31.64	30	249	72.1
		Table Size	25					
	Transmit	Buffer Length	220					
		Buffer Type	P					
Signal Gateway		Rate Conver.			0			
TX	Bus Interface	Buffer Length	150	24.8	29.02	0	200	16.5
		TX Overhead time(ms)	5					
	Bus Controller	H/W Buffer Length	10	0	0	0	199	0
		TX Time (ms)	0.94					
Gateway Performance Summary					89.22	70	1148	
Performance Improvement (%)					-13.1	4.1	-6.6	0

In Case 4, the Buffer Length in Gateway is increased. In this case, only *No. of Dropped Messages* is improved by **4.1** percent. The *Average Message Delay* and *Throughput* are dropped by **13.1** and **6.6** percent respectively. But the *Buffer Utilization* in the PDU Router when compared to all the evaluation cases is the highest.

2. Different Bus Loading and Same Bus Rate

This section evaluates the gateway parameters by setting the bus loading with a high loading and a low loading, and the bus rate with the same value.

Parameter			Value	Remark
RX Bus	Bus Loading (%)		80	When the bus loading is 80% and the bus rate is 500Kbits/s , the msg/sec on the bus is 3389.8 . Suppose 20% of these msg/sec is going to the gateway block is realistic, so each 1.48 ms, an msg is received by the RX bus.
	Bus Rate (Kbits/sec)		500	
TX Bus	Bus Loading (%)		30	When the bus loading is 30% and the bus rate is 500Kbits/s , the msg/sec on the bus is 1271.2 . Each 0.687 ms, an msg is transmitted by the TX bus.
	Bus Rate (Kbits/sec)		500	
RX	Bus Controller	Buffer Length (Msgs)	150	Set the RAM size is 1.2MB , each message has 8 Bytes. (Fixed)
	Bus Interface	RX Overhead Time(ms)	2	(Fixed)
PDU Router	Router	Cycle Time (ms)	2	The Router Cycle time could be 2 ms
		Table Size	25	
	Transmit	Buffer Length (Msgs)	170	Set the RAM size is 1.4MB , each message has 8 Bytes.
		Buffer Type	FIFO	The initial buffer type is set up as FIFO
Signal Gateway		Rate Conversion		
TX	Bus Interface	Buffer Length (Msgs)	150	Set the RAM size is 1.2MB , each message has 8 Bytes. (Fixed)
		TX Overhead time(ms)	5	The Overhead time could be 2~4 ms (Fixed)
	Bus Controller	H/W Buffer Length (Msgs)	10	(Fixed)
		TX Time (ms)	0.687	(Fixed)

Table 10.7: Initial Case Setup 2

Case 1:

Block		Parameter	Value	Avg. Queue Leng.	Avg. Msg Delay (ms)	No. of Dropped Msgs.	Throughput (Msg/sec)	Buffer Utilization (%)
RX	Bus Controller	H/W Buffer Length	150	86.3	26.43	40	500	57.5
	Bus Interface	RX Overhead Time(ms)	2					
PDU_R	Router	Cycle Time (ms)	2	111.6	249	80	249	65.6
		Table Size	25					
	Transmit	Buffer Length	170					
		Buffer Type	F					
Signal Gateway		Rate Conver.			0			
TX	Bus Interface	Buffer Length	150	24.8	18.09	0	200	16.5
		TX Overhead time(ms)	5					
	Bus Controller	H/W Buffer Length	10	0	0	0	199	0
		TX Time (ms)	0.94					
Gateway Performance Summary					293.52	120	1148	
Performance Improvement (%)					0	0	0	0

Case 2:

Block		Parameter	Value	Avg. Queue Leng.	Avg. Msg Delay (ms)	No. of Dropped Msgs.	Throughput (Msg/sec)	Buffer Utilization (%)
RX	Bus Controller	H/W Buffer Length	150	86.3	26.43	40	500	57.5
	Bus Interface	RX Overhead Time(ms)	2					
PDU_R	Router	Cycle Time (ms)	2	111.6	75.76	80	249	65.6
		Table Size	25					
	Transmit	Buffer Length	170					
		Buffer Type	P					
Signal Gateway		Rate Conver.			0			
TX	Bus Interface	Buffer Length	150	24.8	55.58	0	200	16.5
		TX Overhead time(ms)	5					
	Bus Controller	H/W Buffer Length	10	0	0	0	199	0
		TX Time (ms)	0.94					
Gateway Performance Summary					157.77	120	1148	
Performance Improvement (%)					46.2	0	0	0

In this case, the *Average Message Delay* is improved by **46.2** percent, compared to Case 1, but its improvement is not as good as the Case 2 in the first set. This is because of the heavy load in the *RX Configuration Bus* the lower priority messages have to wait a relatively long time before being transmitted.

Case 3:

Block		Parameter	Value	Avg. Queue Leng.	Avg. Msg Delay (ms)	No. of Dropped Msgs.	Throughput (Msg/sec)	Buffer Utilization (%)
RX	Bus Controller	H/W Buffer Length	150	86.3	15.26	40	500	57.5
	Bus Interface	RX Overhead Time(ms)	2					
PDU_R	Router	Cycle Time (ms)	1.5	83.91	27.18	0	332	65.6
		Table Size	25					
	Transmit	Buffer Length	170					
		Buffer Type	P					
Signal Gateway		Rate Conver.			0			
TX	Bus Interface	Buffer Length	150	61.85	38.99	33	199	41.2
		TX Overhead time(ms)	5					
	Bus Controller	H/W Buffer Length	10	0	0	0	198	0
		TX Time (ms)	0.94					
Gateway Performance Summary					81.43	73	1229	
Performance Improvement (%)					48.4	39.2	14.2	0

In the above Case, the results of *Average Message Delay*, *No. of Dropped Messages* and *Throughput* are improved by **48.4**, **39.2** and **14.2** percent respectively. It seems that a small change in the load will cause a big change in the delay and dropped messages.

Case 4:

Block		Parameter	Value	Avg. Queue Leng.	Avg. Msg Delay (ms)	No. of Dropped Msgs.	Throughput (Msg/sec)	Buffer Utilization (%)
RX	Bus Controller	H/W Buffer Length	150	86.3	26.43	40	500	57.5
	Bus Interface	RX Overhead Time(ms)	2					
PDU_R	Router	Cycle Time (ms)	2	122.5	29.95	30	249	65.6
		Table Size	25					
	Transmit	Buffer Length	220					
		Buffer Type	P					
Signal Gateway		Rate Conver.			0			
TX	Bus Interface	Buffer Length	150	24.8	27.68	0	200	16.5
		TX Overhead time(ms)	5					
	Bus Controller	H/W Buffer Length	10	0	0	0	199	0
		TX Time (ms)	0.94					
Gateway Performance Summary					84.06	73	1148	
Performance Improvement (%)					-0.03	0	-14.2	0

In Case 4, the *Average Message Delay* is only slightly dropped by **0.03** percent. No significance difference results when changing the *Buffer Length* and *Cycle Time*. The *Throughput* of this case, however, is dropped by **14.2** percent. This is because the *Buffer Length* is bigger, as some messages are stored in the Buffer.

3. Same Bus Loading and Different Bus Rate

This section evaluates the gateway parameters by setting the bus loading with the same value, and the bus with two different rates.

Parameter			Value	Remark
RX Bus	Bus Loading (%)		80	When the bus loading is 80% and the bus rate is 500Kbits/s , the msg/sec on the bus is 3389.8 . Suppose 20% of these msg/sec is going to the gateway block is realistic, so each 1.48 ms, an msg is received by the RX bus.
	Bus Rate (Kbits/sec)		500	
TX Bus	Bus Loading (%)		80	When the bus loading is 80% and the bus rate is 250Kbits/s , the msg/sec on the bus is 1694.9 . Each 1.899 ms, an msg is transmitted by the TX bus.
	Bus Rate (Kbits/sec)		250	
RX	Bus Controller	Buffer Length (Msgs)	150	Set the RAM size is 1.2MB , each message has 8 Bytes. (Fixed)
	Bus Interface	RX Overhead Time(ms)	2	(Fixed)
PDU Router	Router	Cycle Time (ms)	2	The Router Cycle time could be 2 ms
		Table Size	25	
	Transmit	Buffer Length (Msgs)	170	Set the RAM size is 1.4MB , each message has 8 Bytes.
		Buffer Type	FIFO	The initial buffer type is set up as FIFO
Signal Gateway		Rate Conversion	0.68	
TX	Bus Interface	Buffer Length (Msgs)	150	Set the RAM size is 1.2MB , each message has 8 Bytes. (Fixed)
		TX Overhead time(ms)	5	The Overhead time could be 2~4 ms (Fixed)
	Bus Controller	H/W Buffer Length (Msgs)	10	(Fixed)
		TX Time (ms)	1.899	(Fixed)

Table 10.8: Initial Case Setup 3

Case 1:

Block		Parameter	Value	Avg. Queue Leng.	Avg. Msg Delay (ms)	No. of Dropped Msgs.	Through-put (Msg/sec)	Buffer Utilization (%)
RX	Bus Controller	H/W Buffer Length	150	86.3	23.47	40	500	57.5
	Bus Interface	RX Overhead Time(ms)	2					
PDU_R	Router	Cycle Time (ms)	2	112.1	251.3	80	248	65.9
		Table Size	25					
	Transmit	Buffer Length	170					
		Buffer Type	F					
Signal Gateway		Rate Conver.			0.68			
TX	Bus Interface	Buffer Length	150	24.8	29.9	0	199	16.5
		TX Overhead time(ms)	5					
	Bus Controller	H/W Buffer Length	10	0	0	0	198	0
		TX Time (ms)	0.94					
Gateway Performance Summary					305.36	120	1145	
Performance Improvement (%)					0	0	0	0

Case 2:

Block		Parameter	Value	Avg. Queue Leng.	Avg. Msg Delay (ms)	No. of Dropped Msgs.	Through-put (Msg/sec)	Buffer Utilization (%)
RX	Bus Controller	H/W Buffer Length	150	86.3	23.47	40	500	57.5
	Bus Interface	RX Overhead Time(ms)	2					
PDU_R	Router	Cycle Time (ms)	2	112.1	80.32	80	248	65.9
		Table Size	25					
	Transmit	Buffer Length	170					
		Buffer Type	P					
Signal Gateway		Rate Conver.			0.68			
TX	Bus Interface	Buffer Length	150	24.8	64.36	0	199	16.5
		TX Overhead time(ms)	5					
	Bus Controller	H/W Buffer Length	10	0	0	0	198	0
		TX Time (ms)	0.94					
Gateway Performance Summary					168.83	120	1145	
Performance Improvement (%)					44.7	0	0	0

In this Case, although it is improved by **44.7** percent, the *Average Message Delay* is still a little bit longer, as message transmission is between different transport protocols with different Bus Rates, where there are time costs associated with the rate conversion.

Case 3:

Block		Parameter	Value	Avg. Queue Leng.	Avg. Msg Delay (ms)	No. of Dropped Msgs.	Throughput (Msg/sec)	Buffer Utilization (%)
RX	Bus Controller	H/W Buffer Length	150	86.3	25.31	40	500	57.5
	Bus Interface	RX Overhead Time(ms)	2					
PDU_R	Router	Cycle Time (ms)	1.5	83.63	23.42	0	332	49.2
		Table Size	25					
	Transmit	Buffer Length	170					
		Buffer Type	P					
Signal Gateway		Rate Conver.			0.68			
TX	Bus Interface	Buffer Length	150	61.85	48.4	33	199	41.2
		TX Overhead time(ms)	5					
	Bus Controller	H/W Buffer Length	10	0	0	0	198	0
		TX Time (ms)	0.94					
Gateway Performance Summary					97.81	73	1229	
Performance Improvement (%)					42.1	39.2	7	0

In Case 3, the gateway performance is influenced by the different Bus Rates on both sides of the Gateway ECU model. The results of *Average Message Delay*, *No. of Dropped Messages* and *Throughput* are improved by **42.1**, **39.2** and **7** percent respectively.

Case 4:

Block		Parameter	Value	Avg. Queue Leng.	Avg. Msg Delay (ms)	No. of Dropped Msgs.	Throughput (Msg/sec)	Buffer Utilization (%)
RX	Bus Controller	H/W Buffer Length	150	86.3	18.32	40	500	57.5
	Bus Interface	RX Overhead Time(ms)	2					
PDU_R	Router	Cycle Time (ms)	2	123	34.01	30	248	65.6
		Table Size	25					
	Transmit	Buffer Length	220					
		Buffer Type	P					
Signal Gateway		Rate Conver.			0.68			
TX	Bus Interface	Buffer Length	150	24.8	28.47	0	199	16.5
		TX Overhead time(ms)	5					
	Bus Controller	H/W Buffer Length	10	0	0	0	198	0
		TX Time (ms)	0.94					
Gateway Performance Summary					81.51	73	1145	
Performance Improvement (%)					16.7	0	-7	0

In the above case, the *Average Message Delay* is improved by **16.7** percent. It is different the Case 4 in both Set 1 and Set 2. This difference is due to the message rate conversion operation in the Signal Gateway, which seems to release some of the burden for the *TX Configuration Bus*, so that it has enough time to forward messages onto the bus with less delay.

4. Minimize the dropped messages

From the above three test cases, there always seems to be dropped messages, which is a problem for a gateway. In this section, another test is run by changing the parameters in Bus Configuration RX and TX models. In this set of cases, the initial setup is described in Table 10.9.

Parameter			Value	Remark
RX Bus	Bus Loading (%)		80	When the bus loading is 80% and the bus rate is 500Kbits/s , the msg/sec on the bus is 3389.8 . Suppose 20% of this msg/sec going to the gateway block is realistic, so every 1.48 ms, an msg is received by the RX bus.
	Bus Rate (Kbits/sec)		500	
TX Bus	Bus Loading (%)		80	When the bus loading is 80% and the bus rate is 500Kbits/s , the msg/sec on the bus is 3389.8 . Every 0.94 ms, an msg is transmitted by the TX bus.
	Bus Rate (Kbits/sec)		500	
RX	Bus Controller	Buffer Length (Msgs)	150	Set the RAM size is 1.2MB , each message has 8 Bytes. (Fixed)
	Bus Interface	RX Overhead Time(ms)	2	(Fixed)
PDU Router	Router	Cycle Time (ms)	1.5	The Router Cycle time could be 2 ms
		Table Size	25	
	Transmit	Buffer Length (Msgs)	220	Set the RAM size is 1.4MB , each message has 8 Bytes.
		Buffer Type	Priority	The initial buffer type is set up as FIFO
Signal Gateway		Rate Conversion		
TX	Bus Interface	Buffer Length (Msgs)	150	Set the RAM size is 1.2MB , each message has 8 Bytes. (Fixed)
		TX Overhead time(ms)	5	The Overhead time could be 2~4 ms (Fixed)
	Bus Controller	H/W Buffer Length (Msgs)	10	(Fixed)
		TX Time (ms)	0.94	(Fixed)

Table 10.9: Initial Case Setup 4

Case 1:

Block		Parameter	Value	Avg. Queue Leng.	Avg. Msg Delay (ms)	No. of Dropped Msgs.	Through-put (Msg/sec)	Buffer Utilization (%)
RX	Bus Controller	<i>H/W Buffer Length</i>	150	86.3	26.32	40	500	57.5
	Bus Interface	<i>RX Overhead Time(ms)</i>	2					
PDU_R	Router	<i>Cycle Time (ms)</i>	1.5	83.63	29.93	0	332	72.1
		<i>Table Size</i>	25					
	Transmit	<i>Buffer Length</i>	220					
		<i>Buffer Type</i>	P					
Signal Gateway		<i>Rate Conver.</i>			0			
TX	Bus Interface	<i>Buffer Length</i>	150	61.85	43.78	33	199	16.5
		<i>TX Overhead time(ms)</i>	5					
	Bus Controller	<i>H/W Buffer Length</i>	10	0	0	0	198	0
		<i>TX Time (ms)</i>	0.94					
Gateway Performance Summary					100.03	73	1229	
Performance Improvement (%)					0	0	0	0

In Case 1, inside the gateway appears to work efficiently. There are however, still has dropped messages in both the *RX Bus* and *TX Bus*.

Case 2:

Block		Parameter	Value	Avg. Queue Leng.	Avg. Msg Delay (ms)	No. of Dropped Msgs.	Through-put (Msg/sec)	Buffer Utilization (%)
RX	Bus Controller	H/W Buffer Length	150	45.55	11.85	0	588	57.5
	Bus Interface	RX Overhead Time(ms)	1.5					
PDU_R	Router	Cycle Time (ms)	1.5	124.8	29.82	34	332	72.1
		Table Size	25					
	Transmit	Buffer Length	220					
		Buffer Type	P					
Signal Gateway		Rate Conver.			0			
TX	Bus Interface	Buffer Length	150	61.85	55.03	33	199	16.5
		TX Overhead time(ms)	5					
	Bus Controller	H/W Buffer Length	10	0	0	0	198	0
		TX Time (ms)	0.94					
Gateway Performance Summary					96.7	67	1317	
Performance Improvement (%)					4.5	8.2	7.2	0

In Case 2, the RX Overhead Time is shortened. By making this change, the *Average Message Delay*, *No. of Dropped Messages* and *Throughput* are improved by **4.5**, **8.2** and **7.2** percent. Problems are solved; the *No. of Dropped Messages* in RX is minimized to **0**, but in the *PDU Router*, this result is increased. Thus, this shows that changing the *RX Overhead Time* is not a way to minimize the *No. of Dropped Messages* in the RX Configuration Bus.

Case 3:

Block		Parameter	Value	Avg. Queue Leng.	Avg. Msg Delay (ms)	No. of Dropped Msgs.	Throughput (Msg/sec)	Buffer Utilization (%)
RX	Bus Controller	H/W Buffer Length	200	89.27	15.71	0	500	57.5
	Bus Interface	RX Overhead Time(ms)	2					
PDU_R	Router	Cycle Time (ms)	1.5	83.63	23.87	0	332	72.1
		Table Size	25					
	Transmit	Buffer Length	220					
		Buffer Type	P					
Signal Gateway		Rate Conver.			0			
TX	Bus Interface	Buffer Length	150	61.85	32.87	33	199	16.5
		TX Overhead time(ms)	5					
	Bus Controller	H/W Buffer Length	10	0	0	0	198	0
		TX Time (ms)	0.94					
Gateway Performance Summary					72.45	33	1229	
Performance Improvement (%)					33.5	103	-7.2	0

In Case 3, the *H/W Buffer Length* is increased. By carrying out this change, the *Average Message Delay* and *No. of Dropped Messages* are improved by **33.5** and **103** percent. The *Throughput* however, is decreased by **7.2** percent. In this case, the *No. of Dropped Messages* in RX is minimized to **0**. From the overall Gateway Performance, it is more efficient to change the *RX Overhead Time* than it is to change the *H/W Buffer Length*.

Case 4:

Block		Parameter	Value	Avg. Queue Leng.	Avg. Msg Delay (ms)	No. of Dropped Msgs.	Through-put (Msg/sec)	Buffer Utilization (%)
RX	Bus Controller	H/W Buffer Length	200	89.27	15.71	0	500	57.5
	Bus Interface	RX Overhead Time(ms)	2					
PDU_R	Router	Cycle Time (ms)	1.5	83.63	23.87	0	332	72.1
		Table Size	25					
	Transmit	Buffer Length	220					
		Buffer Type	P					
Signal Gateway		Rate Conver.			0			
TX	Bus Interface	Buffer Length	200	65.95	21.84	0	199	16.5
		TX Overhead time(ms)	5					
	Bus Controller	H/W Buffer Length	10	0	0	0	198	0
		TX Time (ms)	0.942					
Gateway Performance Summary					61.42	0	1229	
Performance Improvement (%)					15.2	100	0	0

Once the *No. of Dropped Messages* in RX is minimized, the next step is to minimize them in TX. In Case 4, the *Buffer Length* in TX is changed to **200**, the *Average Message Delay* and *No. of Message Dropped* are improved by **15.2** and **100** percent. There are no dropped messages in this whole model.

Case 5:

Block		Parameter	Value	Avg. Queue Leng.	Avg. Msg Delay (ms)	No. of Dropped Msgs.	Throughput (Msg/sec)	Buffer Utilization (%)
RX	Bus Controller	H/W Buffer Length	200	89.27	15.59	0	500	57.5
	Bus Interface	RX Overhead Time(ms)	2					
PDU_R	Router	Cycle Time (ms)	1.5	83.63	13.3	0	332	72.1
		Table Size	25					
	Transmit	Buffer Length	220					
		Buffer Type	P					
Signal Gateway		Rate Conver.			0			
TX	Bus Interface	Buffer Length	100	41.33	15.69	0	249	16.5
		TX Overhead time(ms)	3					
	Bus Controller	H/W Buffer Length	10	0	0	0	248	0
		TX Time (ms)	0.94					
Gateway Performance Summary					44.58	0	1329	
Performance Improvement (%)					27.4	0	8.1	0

In Case 5, the *TX Overhead Time* improves, while the *Average Message Delay* and *Throughput* are improved by **27.4** and **8.1** percent respectively.

10.7 CONCLUSION

Testing the current simulation model allows the design to be verified and validated against the AUTOSAR defined in-vehicle network gateway specification, which includes interfaces, communication bus and gateway ECU.

The verification testing has shown that all forms of message transmission and reception within vehicle networks are correctly simulated without errors.

The validation testing has shown that the whole in-vehicle network model is correctly simulated according to the AUTOSAR specification. Validation testing, also clearly shows that the buffer mechanism plays a very important role when designing a vehicle network gateway. This testing proves conclusively that when designing an in-vehicle network gateway, the critical factor for consideration is the unpredictable message delays, which depend on the bus loading and buffer mechanism within the gateway.

10.8 REFERENCES

JERRY BANKS, J. S. C., BARRY L. NELSON, DAVID M. NICOL (2000A)

DISCRETE-EVENT SYSTEM SIMULATION PRENTICE HALL.

MATHWORKS, T. (1997A) MATLAB AND SIMULINK USER'S GUIDES. THE

MATHWORKS, INC.

MATHWORKS, T. (2006) SIMEVENTS USER'S GUIDES, THE MATHWORKS, INC.

11.1 RESEARCH SUMMARY

The current research investigated the AUTOSAR defined gateway specification by using Matlab & Simulink (SimEvents) to build a gateway ECU simulator with all essential in-vehicle network components attached.

This research began with a very broad investigation on in-vehicle networks, which included vehicle network protocols, network design, management and network gateway design. This was followed by the exploration of different methodologies, currently used in software development and especially in the automotive industry. Other areas that were investigated in depth include simulation technologies, such as Queuing Theory and the MATLAB/Simulink and SimEvents environment.

This research then examined a very specific objective, which was to investigate the AUTOSAR defined gateway specification. Meanwhile, different in-vehicle network components, such as ECUs and Communication Buses, as well as concepts, such as Event-triggered and Time-triggered were also considered.

The main task of this research was to simulate an in-vehicle network gateway ECU with all necessary network components. This gateway ECU simulation executes the messages dynamically generated from the application ECUs across the communication bus. For the communication bus, each single message transmit time was collected from a hardware board under different bus loadings generated by a software package. Using the gateway ECU simulation, different parameters were changed and the results evaluated to optimize the gateway performance.

CONCLUSION

The methodology used in developing this gateway ECU for this research was a model-based design applied in a V-process model via a series of rapid prototyping ideas.

11.2 RESEARCH CONCLUSION

This research aimed to answer key questions identified at the beginning of the research process. These questions have been fully researched and investigated during this 2 year process.

Research Question: Which aspects of an AUTOSAR gateway configuration have a significant impact on gateway performance?

At the beginning of this research, an in-vehicle network was considered a small local area network which is suitable especially for use in real-time distributed control systems. Data transmission in a distributed control system is influenced significantly by the intensity and distribution of message traffic in the network. Data transmission is subject to time-varying delays due to the latency of messages. In addition, messages may be lost due to buffer saturation in the receiving stations and, therefore, have to be retransmitted. In order to solve these problems, Queuing Theory was applied to the in-vehicle network gateway.

It was found that by analysing the Queuing Theory, some idea about the relationship among message delays, priorities, status (bus loading) of the system are invaluable when choosing optimum gateway design parameters.

The optimal gateway configuration was achieved by creating different evaluation cases. This in turn was achieved by changing different critical parameters set up inside the gateway simulation model and comparing the results displayed from each simulation run. These results clearly show that Buffer Mechanism plays a very important role when designing an in-vehicle network gateway. Shorter buffer lengths are preferable for short message delays, but the buffer must be large enough so as not to cause dropped messages. The Priority queue also gave better performance compared to the FIFO queue in the Gateway Model.

Research Question: Is the Matlab/Simulink and SimEvents a feasible environment to model and simulate the AUTOSAR defined in-vehicle network gateway system?

The Matlab/Simulink and SimEvents environment proved very applicable for developing and testing the AUTOSAR defined in-vehicle network gateway system.

It was determined that, the SimEvents package of Simulink combined with the Queuing Theory and the AUTOSAR defined gateway feature perfectly.

Research Question: Can a gateway simulation model be used effectively to optimise gateway performance?

From the results of this research and from discussions with experts in the in-vehicle network research domain, and experts from AUTOSAR, a comprehensive gateway simulation model is a valuable tool for optimising a gateway's performance.

This simulated gateway model could be incorporated into hardware in the loop automotive design systems which utilises – D Space hardware which is very good at simulating real time events of hardware components on the engine thereby giving a even more realistic data output which would be most desirable to designers in the automotive field today.

11.3 AREAS FOR FURTHER RESEARCH

In the Optimisation of Gateway Performance section of the Testing chapter, the evaluation cases are more concentrated on general gateway parameters. However, the design of the gateway has to follow the technology development in automotive industry. Therefore there are some very worthwhile avenues for further research. These include:

- In vehicle network gateways, there are lots of different parameters that can be evaluated. In the current research, only some critical ones are used as an example to demonstrate the simulation model. Suggestion by some members of the SAE (Society of Automotive Engineers) include researching different protocol conversions in signal gateway, DMA approach and arbitration mechanisms.
- There is more and more functionality embedded into the in-vehicle network gateways, such as the information for vehicle manufacturers, vehicle owners and vehicle drivers, which needs to be protected by gateways. The configuration of gateways must become more intelligent to manage such data. Further research could examine the impact of this extra data management on gateway performance.

PART FIVE

APPENDICES



APPENDIX A BIBLIOGRAPHY

12 BIBLIOGRAPHY

AUTOMOTIVE ELECTRONICS, V. (2004) AUTOMOTIVE ELECTRONICS: MODEL-BASED DEVELOPMENT WITH VIRTUAL PROTOTYPES. VAST SYSTEM TECHNOLOGY.

AUTOSAR, AUTOMOTIVE OPEN SYSTEM ARCHITECTURE.

AUTOSAR (2006) SPECIFICATION OF PDU ROUTER, AUTOSAR GbR.

AXELSSON, J. (1999) HOLISTIC OBJECT-ORIENTED MODELLING OF DISTRIBUTED AUTOMOTIVE REAL-TIME CONTROL APPLICATIONS. VOLVO TECHNOLOGICAL DEVELOPMENT CORP

BJÖRN WESTMANN (2006) DESIGNING NETWORKS VS. TESTING NETWORKS. ED., MENTOR GRAPHICS.

BOSCH (1991) CAN SPECIFICATION VERSION 2.0. STUTTGART, ROBERT BOSCH GMBH.

BROWN, B. (1996-2000) NETWORKING FUNDAMENTALS NETWORK TOPOLOGY.

CIA (2004) CAN-BASED HIGHER-LAYER PROTOCOLS AND PROFILES. ED., CAN IN AUTOMATION.

CIA (2007) CONTROLLER AREA NETWORK (CAN) -- PROTOCOL. ED., CAN IN AUTOMATION.

E.DILGER, T. F., B.MULLER (1998) THE X-BY-WIRE CONCEPT: TIME TRIGGERED INFORMATION EXCHANGE AND FAIL SILENCE SUPPORT BY NEW SYSTEM SERVICES. SAE.

EASIS (2005A) CONCEPTUAL HARDWARE ARCHITECTURE SPECIFICATION. VERSION 1.0 ED., ELECTRONIC ARCHITECTURE AND SYSTEM ENGINEERING FOR INTEGRATED SAFETY SYSTEMS.

EASIS (2005B) CONCEPTUAL HARDWARE ARCHITECTURE SPECIFICATION, VERSION 1.0. EASIS - ELECTRONIC ARCHITECTURE AND SYSTEM ENGINEERING FOR INTEGRATED SAFETY SYSTEMS

ETTSCHBERGER, K. (2001) CONTROLLER AREA NETWORK. ED. WEINGARTEN, GERMANY, IXXAT PRESS.

F.HARTWICH, B. M., T. FUHRER AND R. HUGEL (2000) CAN NETWORK WITH TIME TRIGGERED COMMUNICATION. CIA (CAN IN AUTOMATION). ROBERT BOSCH GMBH.

FLEXRAY-CONSORTIUM (2000) FLEXRAY SPECIFICATION VERSION 2.1. FLEXRAY-CONSORTIUM.

FUJITSU (2006) NEXT GENERATION CAR NETWORK -- FLEXRAY. ED., FUJITSU MICROELECTRONICS (SHANGHAI) CO., LTD.

H. KOPETZ, G. G. (1993) TTP – A TIME TRIGGERED PROTOCOL FOR FAULT-TOLERANT REAL TIME SYSTEMS. PROC. 23RD IEEE INTERNATIONAL SYMPOSIUM ON FAULT TOLERANT COMPUTING. IEEE PRESS (PG 524-532).

HARVEY GOULD, J. T., WOLFGANG CHRISTIAN (2006) AN INTRODUCTION TO COMPUTER SIMULATION METHODS: APPLICATIONS TO PHYSICAL SYSTEMS ADDISON WESLEY.

HEURUNG, T. IN-VEHICLE NETWORK DESIGN METHODOLOGY. MENTOR GRAPHICS

IBM (1950s) FORTRAN, INTERNATIONAL BUSINESS MACHINES CORPORATION (IBM).

INFINEON INFINEON C166 MCU. INFINEON.

INS (2002) BASIC NETWORK DESIGN METHODOLOGY. INS WHITEPAPER

ISO (1993) ROAD VEHICLE – INTERCHANGE OF DIGITAL INFORMATION – CONTROLLER AREA NETWORK. HUTING VERLAG, HEIDELBERG, GERMANY, INTERNATIONAL ORGANIZATION FOR STANDARDIZATION.

ISO (1994) INFORMATION TECHNOLOGY – OPEN SYSTEM INTERCONNECTION – BASIC REFERENCE MODEL – CONVENTIONS FOR THE DEFINITION OF OSI SERVICES. HUTING VERLAG, HEIDELBERG, GERMANY, INTERNATIONAL ORGANIZATION FOR STANDARDIZATION.

JACK DONGARRA, J. B., CLEVE MOLER, PETE STEWART (1980s) LINPACK

JACOBSEN, M. (2006) EXAMPLES OF QUEUEING MODELS BIRKHÄUSER BOSTON.

JERRY BANKS, J. S. C., BARRY L. NELSON, DAVID M. NICOL (2000A) DISCRETE-EVENT SYSTEM SIMULATION PRENTICE HALL.

K.J, K. U. A. N. (1995) OPEN SYSTEM AND INTERFACES FOR DISTRIBUTED ELECTRONICS IN CARS (OSEK). SAE (SOCIETY OF AUTOMOTIVE ENGINEERS). DETROIT, USA, SAE

LAW, A. M., AND W.D. KELTON (2000) SIMULATION MODELLING AND ANALYSIS
MCGRAW-HILL EDUCATION - EUROPE.

LEMIEUX, J. (2001A) PROGRAMMING IN THE OSEK/VDX ENVIRONMENT, CMP
BOOKS; PAP/CDR EDITION

LEMIEUX, J. (2001B) PROGRAMMING IN THE OSEK/VDX ENVIRONMENT CMP
BOOKS.

MATHWORKS, T. (1984) MATLAB, THE MATHWORKS.

MATHWORKS, T. (1997A) MATLAB AND SIMULINK USER'S GUIDES. THE
MATHWORKS, INC.

MATHWORKS, T. (2006) SIMEVENTS USER'S GUIDES, THE MATHWORKS, INC.

MORGAN, G. (2006) AUTOSAR -- CONTENT, METHODS, AND APPLICATIONS.
LIVEDEVICES, ETAS GROUP.

MORRISSEY, D. (2005) OSEK/VDX PRESENTATION. SAE (SOCIETY OF
AUTOMOTIVE ENGINEERS). AUTOMOTIVE CONTROL GROUP, WATERFORD INSTITUTE OF
TECHNOLOGY.

MUSSELMAN, K. J. (1998) GUIDELINES FOR SUCCESS, JOHN WILEY & SONS INC.

NAYLOR, T. H., J.L.BALINFY, D.S.BURDICK, AND K.CHU (1966) COMPUTER
SIMULATION TECHNIQUES, JOHN WILEY, NEW YORK.

NICOLAS NAVET, Y. S. (JUNE 2005) TRENDS IN AUTOMOTIVE COMMUNICATION
SYSTEMS. THE IEEE.

OSEK/VDX (2004A) OSEK/VDX NETWORK MANAGEMENT SPECIFICATION. CONCEPT
AND APPLICATION PROGRAMMING INTERFACE. OSEK VDX PORTAL.

OSEK/VDX (2004B) OSEK/VDX NETWORK MANAGEMENT SPECIFICATION. CONCEPT
AND APPLICATION PROGRAMMING INTERFACE. OSEK VDX PORTAL.

PEAK PCAN EXPLORE 3. PEAK SYSTEM.

PRITSKER, A. A. B. (1995) INTRODUCTION TO SIMULATION AND SLAM II JOHN WILEY
& SONS INC.

RICHARDS, P. (2002) A CAN PHYSICAL LAYER DISCUSSION. ED., MICROCHIP
TECHNOLOGY INC.

RU, W. Q. (1995) C LANGUAGE AND COMPUTER SIMULATION, 北京航空航天大学出
版社.

S.BROWN, C. (2006) IN-VEHICLE NETWORK DESIGN METHODOLOGY. MENTOR
GRAPHICS

SHANNON, R. E. (1975) SYSTEMS SIMULATION: THE ART AND SCIENCE, PRENTICE HALL

SMITH, P. (2005) AUTOMOTIVE GATEWAYS SPEARHEAD IN-CAR NETWORK INTEGRATION. ED., AUTOMOTIVE DESIGNLINE.

STALLINGS, W. (2002) COMPUTER ORGANIZATION AND ARCHITECTURE: DESIGN FOR PERFORMANCE, PRENTICE HALL.

THOMAS FUHRER, B. M. TIME TRIGGERED COMMUNICATION ON CAN (TIME TRIGGERED CAN -- TTCAN). CIA (CAN IN AUTOMATION).

TORIN, Ö. A. A. J. (2000) FAIL-SILENT COMPUTER NODE. DEPARTMENT OF COMPUTER ENGINEERING, CHALMERS.

U., K. (1990) DISTRIBUTED REAL-TIME PROCESSING IN AUTOMOTIVE NETWORKS. SAE (SOCIETY OF AUTOMOTIVE ENGINEERS). DETROIT, USA, SAE

WEISSTEIN, E. W. MARKOV PROCESS WOLFRAMMATHWORLD.

WILKINSON, J. (1972-1973) EISPACK

APPENDIX

B

APPENDIX B SAE PAPER

13 SAE PAPER

Using Simulation for Designing In-vehicle Network Gateways

Weida Zhu

Waterford Institute of Technology, Ireland

Brendan Jackman

Waterford Institute of Technology, Ireland

Copyright © 2007 SAE International

ABSTRACT

The network is becoming the development focus for the in-vehicle electronic system. Network buses are used to improve communication between ECUs and to reduce the wiring costs. In-vehicle network buses, such as CAN, LIN, FlexRay, have become the central technique for sharing sensor data among vehicle ECUs.

Gateways are a critical factor in vehicle network design with applications requiring the use of several networking standards. There are lots of networking protocols to choose from – each with advantages and disadvantages. No one protocol satisfies the requirements of all automotive applications. There is a need to consolidate data from these networks using de-centralized processing. As such, a gateway is used as a central hub to interconnect and process data from a vehicle's embedded networks. A gateway is composed of several automotive networking interfaces such as CAN, LIN and FlexRay in addition to embedded micro-controllers and peripheral functions.

Meanwhile, simulation has becoming an efficient development tool used in the modern automotive industry. This paper will suggest a simulation solution for designing in-vehicle network gateways.

INTRODUCTION

The In-vehicle network today has become very complex. Generally it is classified to three types [1]:

Class A Multiplexing: is used for convenience features (entertainment, audio, trip computer, etc.); does not require high bandwidth;

Class B Multiplexing: is used for general information transfer (instrument cluster, vehicle speed, legislated emissions data, etc.), requires medium speed;

Class C Multiplexing: requires high bandwidth, reliability, and high data integrity.

Gateways are required to exchange data between these different vehicle networks. Gateways typically exchange messages between connected networks based solely on the destination and priority of the messages. Such gateways can result in unpredictable message delays depending on the network loading and vehicle operating conditions.

Simulink is a design environment based on Matlab [2] for modeling, simulating, and analyzing dynamic systems. It has a comprehensive ability to model any system represented by math, including linear and nonlinear systems, time-driven and event-driven systems. Simulink provides plenty of functional models for different domains to build a whole dynamic system without writing any code.

In the latest Matlab version, Simulink provides an extensive tool called SimEvents for modeling and simulating discrete-event systems using queues and servers. SimEvents allows users to build a discrete-event simulation model in Simulink to simulate the passing of entities through a network of queues, servers, gates, and switches based on events. SimEvents and Simulink provide an integrated environment for modeling hybrid dynamic systems containing continuous-time, discrete-time, and discrete-event components.

VEHICLE NETWORK TOPOLOGY

A network topology is a physical layout, which connects different nodes of a communication network. For an in-vehicle network, there are various methods for connection between different protocol nodes, but only a few methods can work properly in vehicle networks. To decide on a network topology, the chosen protocol standard and the physical interfaces have to be considered.

Figure 1 illustrates a typical vehicle network topology, showing the use of a gateway to connect different network types.

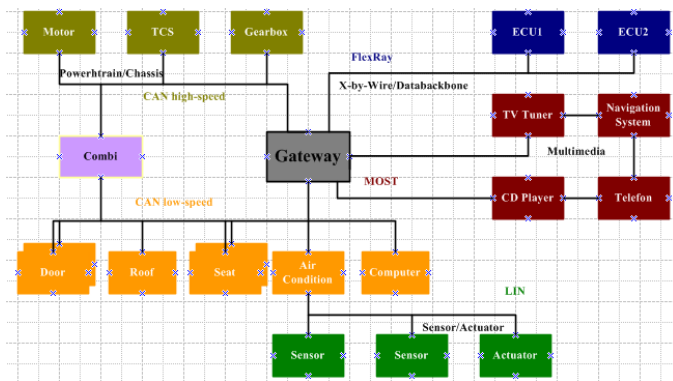


Figure 1: Networking of modern vehicles with gateway

VEHICLE GATEWAY STRUCTURE

AUTOSAR has recently released a detailed document on in-vehicle network gateway. Figure 2 shows the basic gateway structure defined by AUTOSAR [3].

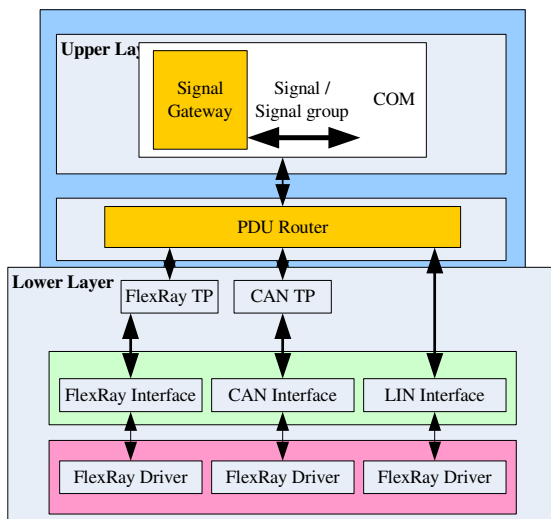


Figure 2: AUTOSAR gateway structure

COM defined by OSEK/VDX [4] is a method of exchanging data between different tasks on a single ECU and between multiple ECUs over a network. COM is an asynchronous communication model, where the application is not required to wait for a message transfer before it resumes processing and it is not blocked if a message is not available when it is requested. COM defines a number of notification mechanisms that assist the application in understanding when a message has been sent or received.

Each message defined for an application can have only one sender within the system, but one or many receivers

can receive it. These receivers can be tasks that reside on the same or a different ECU.

Signal-based Gateway is inside of the COM. Signals and signal groups are stamped with unique static names by signal router. A routing table mechanism is used to store all the signals or signal groups' information including their names, destination, etc. Its main tasks are:

- Packing signals or signal groups (Complex Data Types) from applications into message units (PDU).
- Unpacking signals or signal groups from message units (PDU) to applications.

PDU Router is located between Upper Layer COM and Lower Layer. Its main tasks are:

- Providing a transport platform for messages with different protocol formats on the Lower Layer
- Providing a transport platform for different functional networks on the Lower Layer
- Providing a transport platform for packing/unpacking messages into Upper Layer.

Lower Layer includes Transport Protocols, Protocol Interface, and Protocol Drivers, which are close to the physical layer.

SIMULATION OBJECTIVE

Today, functions in a vehicle ECU are increasing, which is causing the number of ECUs in a vehicle to increase also. The more ECUs that are connected onto the networks, the more burdens a gateway will have. Many OEMs have to consider lots of variations when designing vehicle gateways. In order to design a reliable gateway, those variations must be accounted for during testing. Using the prototyping approach, it is quite hard to build enough physical prototypes to perform adequate testing, even with the most critical variations. Simulating a virtual prototype of a gateway is a more effective solution for verifying its performance.

SIMULATION MODEL

The key elements for simulating an in-vehicle gateway are Upper Layer (COM with signal-based gateway included), PDU Router and Lower Layer components (protocol interfaces, protocol drivers, etc). In a simple but typical gateway model, shown in figure 3, messages arrive from time to time and join a queue, or waiting line, and are eventually served, and finally sent onto the bus.

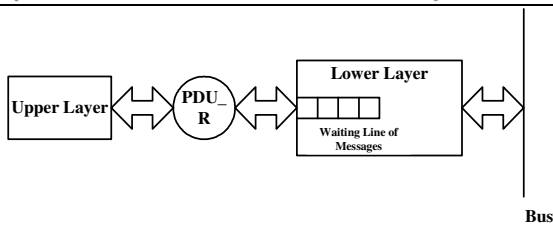


Figure 3: Typical gateway model

According to the specification defined by AUTOSAR, in-vehicle gateways can be simulated separately by Reception and Transmission operations.

RECEPTION SEQUENCE DIAGRAM

There are two types of reception models, non-TP_PDU_RX and TP_PDU_RX.

During reception, the PDU Router works as a server, which is triggered by a RX indication is received from a Lower Layer module. If a RX indication is from the transport protocol module and the first frame of the whole message is received, then the PDU router will ask the Upper Layer to provide a receive buffer. Once the last frame of the whole message is received, the PDU router will send an indication to the Upper Layer informing it of the end of the RX process.

Figure 4 shows the TP_PDU_RX sequence. During the transport protocol RX, multiple PDUs may need to be received, so it is necessary to execute this sequence into a loop for each received PDU. The end of RX will be indicated only after all the PDUs are received.

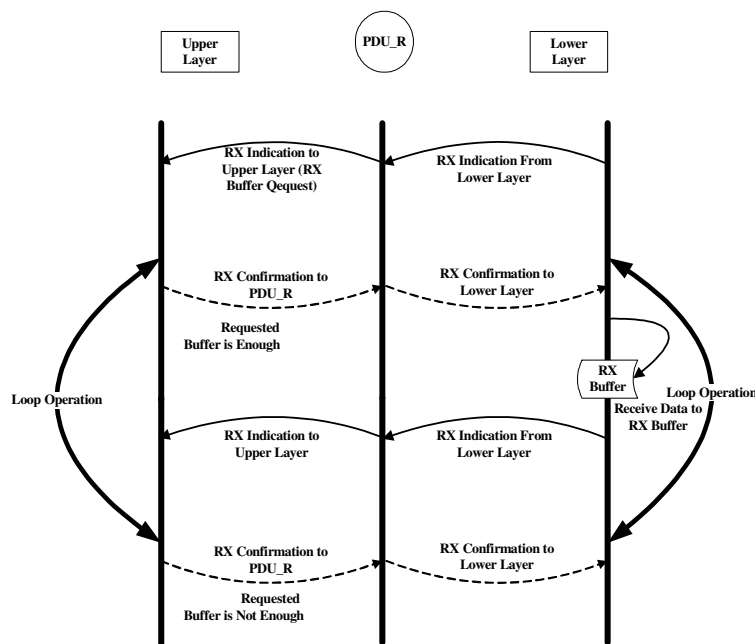


Figure 4: TP_PDU_RX

Figure 5 shows the non-TP_PDU_RX sequence.

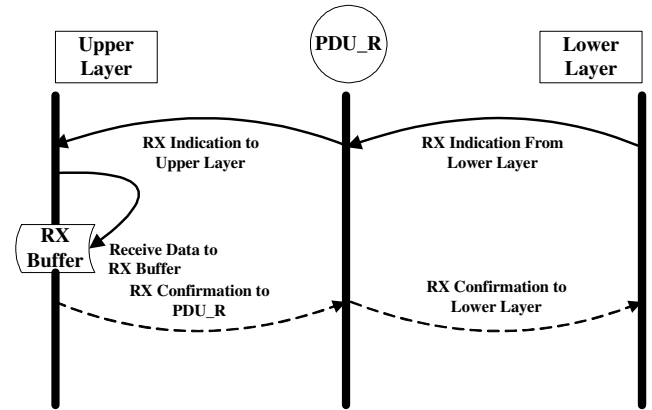


Figure 5: Non-TP_PDU_RX

TRANSMISSION SEQUENCE DIAGRAM

There are two transmission models: non-TP_PDU_TX and TP_PDU_TX.

In the transmission process, the PDU router works as a server forwarding PDUs from Upper Layer to Lower Layer according to the PDU identifiers predefined by the users. The non-TP_PDU_TX is used here as an example.

In the non-TP_PDU_TX, PDUs transmission can be divided into triggered or non-triggered.

The triggered transmission procedure, shown in figure 6, is:

- PDU router gets a transmit request from the Upper Layer,
- PDU router sends the request to the Lower Layer, but does not forward the PDU until it is triggered by the Lower Layer,
- Once it gets the trigger, the PDU router will ask the Upper Layer to provide a transmit buffer, then send the PDU,
- Once the transmission has ended, the Lower Layer will inform a transmit confirmation to the Upper Layer.

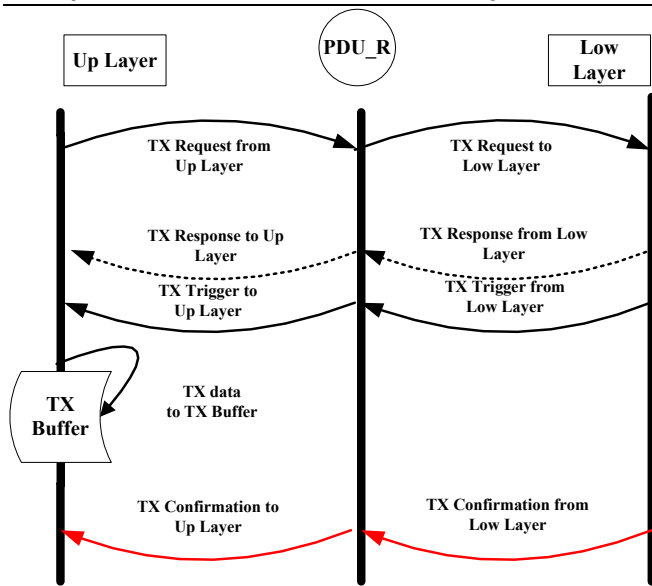


Figure 6: Non-TP_PDU_TX with trigger

The non-triggered transmission procedure, shown in figure 7, is:

- PDU router gets a transmit request from the Upper Layer,
- PDU router sends the request to the Lower Layer, and forwards the PDU with the request,
- During this procedure, the Lower Layer provides the TX buffer,
- Once the transmission has ended, the Lower Layer will inform a transmit confirmation to the Upper Layer.

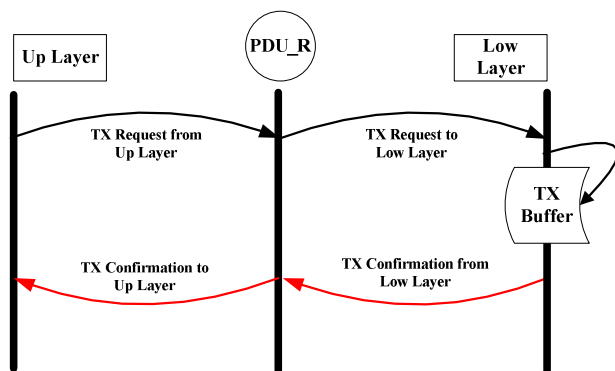


Figure 7: Non-TP_PDU_TX without trigger

SIMULATION MODEL CONFIGURATION

As described previously, SimEvents allows users to build a discrete-event simulation model in Simulink, to simulate the passing of entities through a network of queues, servers, gates, and switches based on events. As such SimEvents can be used to model in-vehicle gateway features.

Here using two transmission examples, Non TP_PDU_TX without Triggered Transmission and Non TP_PDU_TX with Triggered Transmission, it is explained how Upper Layer, PDU router, and Lower Layer are configured inside a gateway using SimEvents.

Figure 8 is the Upper Layer of the Multicast Non TP-PDU-TX module. In this figure, above the line is the TX transmission part, including triggered and Non-triggered. The Set Msg Attributes block assigns an identifier for each message generated by the Msg Generator block. The Msg Replicate for Triggered and Non Triggered block outputs the coming message, one is for Non-triggered, and one is for Triggered. As described in the AUTOSAR specification, the Upper Layer provides the FIFO buffers for Triggered TX. To make Triggered TX work, an Enable Gate block is used to work with the Lower Layer. Below the line is the TX confirmation part in the Upper Layer, a FIFO buffer is used for receiving coming TX confirmations.

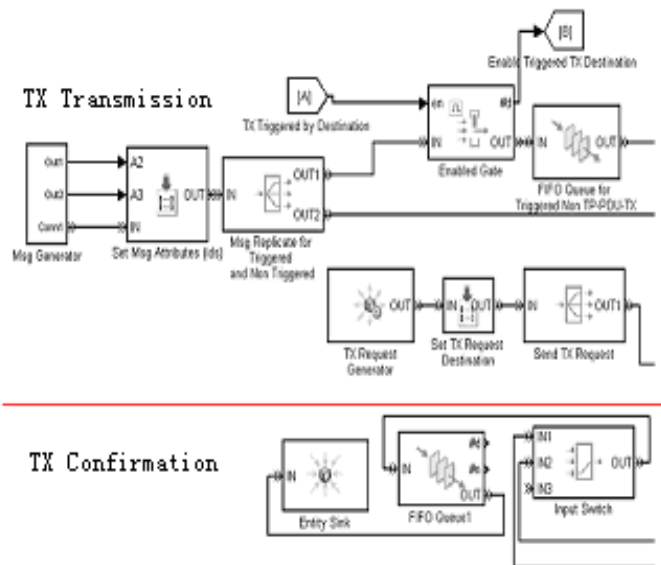


Figure 8: Upper Layer of the Multicast Non TP-PDU-TX module

Figure 9 is the PDU Router of the Multicast Non TP-PDU-TX module. In this figure, above the line, blocks are used for transferring messages from the Upper Layer to the Lower Layer. Below the red line, blocks are used for transferring the TX request from the Upper Layer to the Lower Layer.

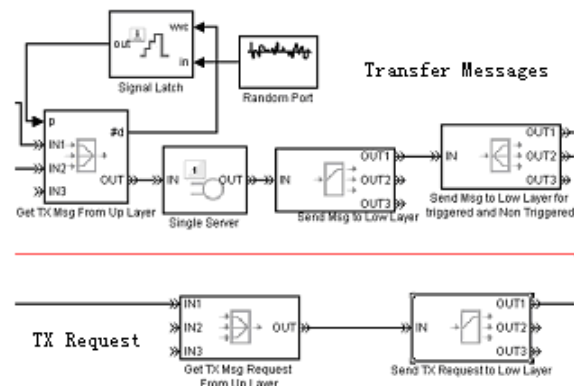


Figure 9: PDU Router of the Multicast Non TP-PDU-TX module

Figure 10 is the Lower Layer of the Multicast Non TP-PDU-TX module. In this figure, triggered transmission above the first line can get the messages from the Upper Layer. Blocks between the first and second lines are used to get messages without triggers.

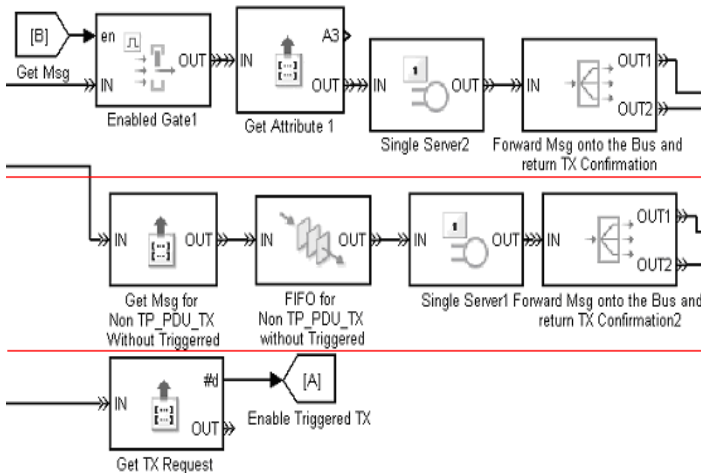


Figure 10: Lower Layer of the Multicast Non TP-PDU-TX module

PDU_Router has to be triggered to forward message data from the Upper Layer to Lower Layer.

EVALUATION OF SIMULATION

As previously mentioned, a gateway has to deal with large numbers of messages between the Upper Layer and Lower Layer, and between different transport protocols. Parameters such as buffer sizes, buffer types and busload can be changed to observe the average message transmission delay time within the gateway simulation model.

SimEvents provides timer blocks for the user to observe how long each entity takes to advance from one block to

another. The timer can be used in the gateway simulation model to get the message processing time within a gateway. In the Non TP_PDU_TX simulation model, two Timer blocks are used, Start Timer block and Read Timer. Once a message is generated, the Start Timer block will start timing. The transmission time can be read after the message passes through the Read Timer block.

EVALUATION OF ALTERNATIVE BUFFER SIZES

1. Setup a single FIFO buffer

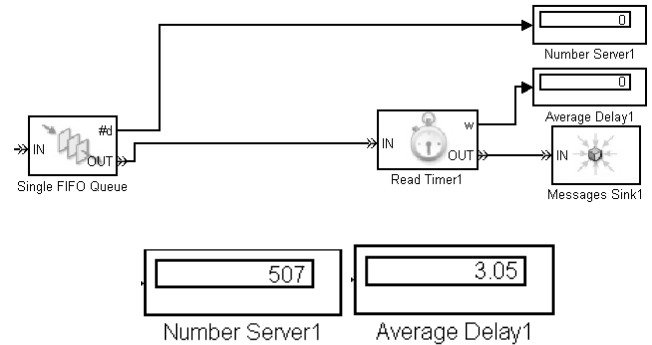


Figure 11: Single FIFO No. Served and Average Delay

2. Setup three FIFO buffers

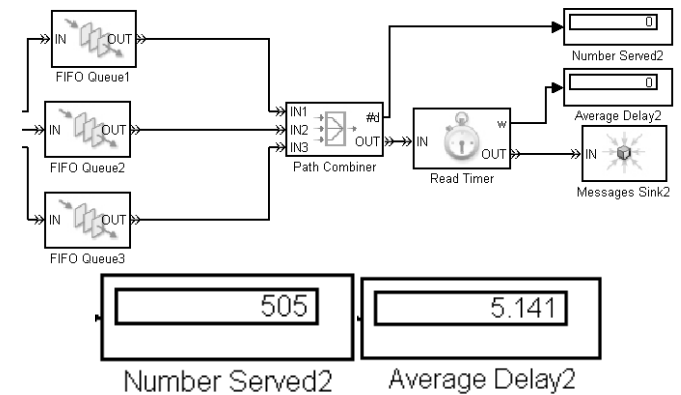


Figure 12: multiple FIFOs No. Served and Average Delay

Figures 11 and 12 show that if using a FIFO buffer mechanism (within the same simulation time), the smaller the buffer size, the smaller the average delay time, and the more messages it can serve.

EVALUATION OF ALTERNATIVE BUFFER MECHANISMS

1. Setup FIFO buffer mechanism

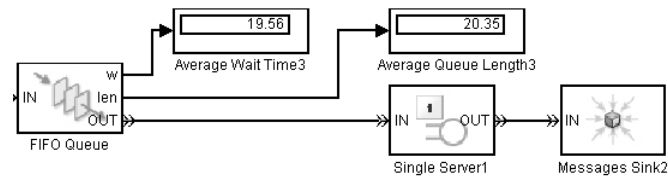


Figure 13: FIFO mechanism

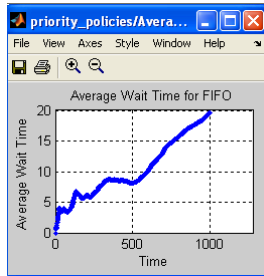


Figure 14: FIFO Average Delay

2. Setup Priority buffer mechanism

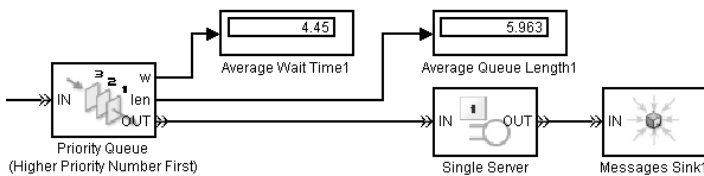


Figure 15: Priority Mechanism

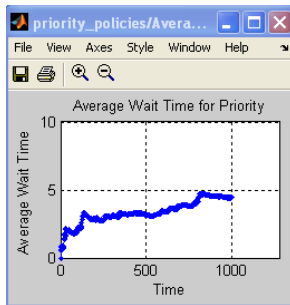


Figure 16: Priority Average Delay

Figures 14 and 16 show that if using a priority buffer mechanism (within the same simulation time), the average delay time will be shorter than if using a FIFO buffer mechanism.

EVALUATION OF BUS LOADING

Figure 17 shows a Bus Loading (250Kbit/Sec) simulation model, Bus Load for 250Kbit/Sec block generates different busloads into SimTime for 250Kbit/Sec S-function block, then Sim Time block outputs the time, which are obtained from the Infineon C167CS [5]. The Single Server block uses the Sim Time as its service time to process each message.

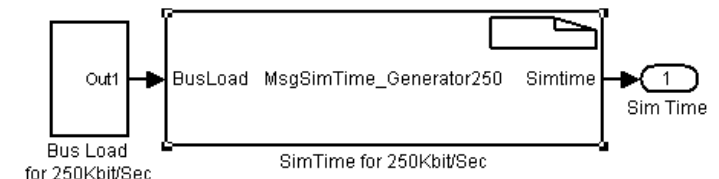


Figure 17: Generate Simulation Time

1. Bus Load is 30.8%

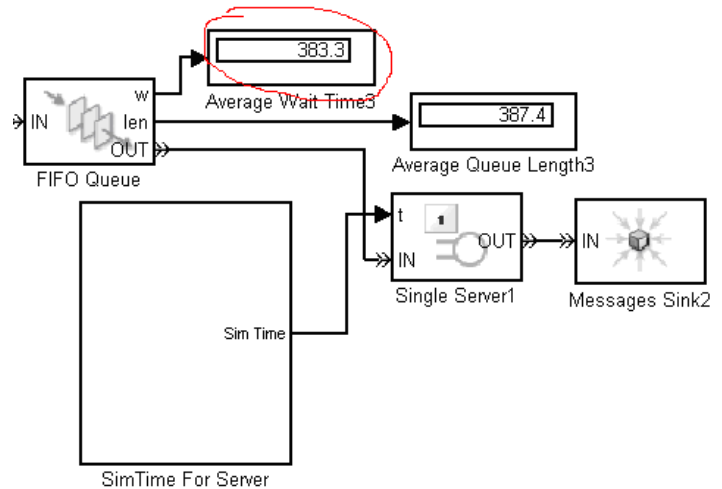


Figure 18: Low Bus Load Average Wait Time

2. Bus Load is 40.8%

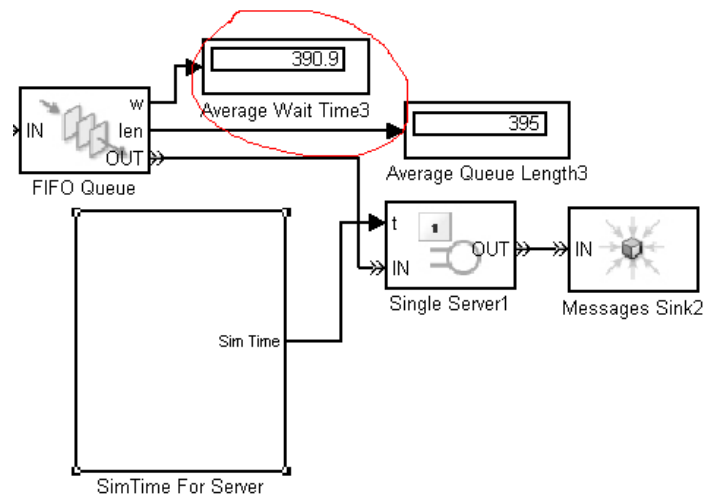


Figure 19: High Bus Load Average Wait Time

Figures 18 and 19 show that the higher the busloads, the more average wait time each message has.

CONCLUSION

From the given model, it shows clearly that buffer mechanism plays a very important role when designing a vehicle network gateway. As mentioned previously,

when designing an in-vehicle network gateway, the critical factor to be considered is the unpredictable message delays, which depends on the bus loading and buffer mechanisms within the gateway. If engineers use the simulation approach, the design life cycle will be obviously decreased for verifying different situations such as bus speed, electronic components, and gateway. By using the virtual approach, simulation can give more general results, and easy configuration.

REFERENCES

1. [1]SAE in-vehicle networks classification, SAE <http://www.sae.org/>
2. [2]Matlab, The MathWorks, <http://www.mathworks.com>
3. [3]AUTOSAR Specifications, AUTOSAR, <http://www.autosar.org>
4. [4]OSEK/VDX, <http://www.osek-vdx.org>
5. [5] Infineon C167CS Data Sheet, Infineon Technologies, <http://www.infineon.com>

CONTACT

Weida Zhu B.Sc.

Weida graduated with a honor degree 2.1 in applied computing in Waterford Institute of Technology. He is currently doing a postgraduate in vehicle network gateway under the supervision of Brendan Jackman.

Email: wzhu@wit.ie

Brendan Jackman B.Sc. M.Tech.

Brendan is the founder and leader of the Automotive Control Group at Waterford Institute of Technology, where he supervises postgraduate students working on automotive software development, diagnostics and vehicle networking research. Brendan also lectures in Automotive Software Development to undergraduates on the B.Sc. in Applied Computing Degree at Waterford Institute of Technology. Brendan has extensive experience in the implementation of real-time control systems, having worked previously with Digital Equipment Corporation, Ireland and Logica BV in The Netherlands.

Email: bjackman@wit.ie

DEFINITIONS, ACRONYMS, ABBREVIATIONS

COM: Communication

ECU: Electronic Control Unit

Non-TP_PDU_RX: Non-Transport Protocol PDU Reception

Non-TP_PDU_TX: Non-Transport Protocol PDU Transmission

OEM: Original Equipment Manufacture

PDU: Protocol Data Unit

RX: Receive

TP: Transport Protocol

TP_PDU_RX: Transport Protocol PDU Reception

TP_PDU_TX: Transport Protocol PDU Transmission

TX: Transmit

