

Marco Di Natale · Haibo Zeng
Paolo Giusto · Arkadeb Ghosal

Understanding and Using the Controller Area Network Communication Protocol

Theory and Practice



Springer

Understanding and Using the Controller Area Network Communication Protocol

Marco Di Natale • Haibo Zeng • Paolo Giusto
Arkadeb Ghosal

Understanding and Using the Controller Area Network Communication Protocol

Theory and Practice

Marco Di Natale
Scuola Superiore S. Anna
RETIS Lab.
Pisa 56124, Italy

Paolo Giusto
General Motors Corporation
Palo Alto, CA 94306, USA

Haibo Zeng
McGill University
Montreal, Quebec, Canada H3A 2A7

Arkadeb Ghosal
National Instruments
Berkeley, CA 94704, USA

ISBN 978-1-4614-0313-5 e-ISBN 978-1-4614-0314-2
DOI 10.1007/978-1-4614-0314-2
Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2011944254

© Springer Science+Business Media, LLC 2012

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This book is the result of several years of study and practical experience in the design and analysis of communication systems based on the Controller Area Network (CAN) standard. CAN is a multicast-based communication protocol characterized by the deterministic resolution of the contention, low cost, and simple implementation. The CAN [16] was developed in the mid 1980s by Bosch GmbH, to provide a cost-effective communications bus for automotive applications. Today it is widely used also in factory and plant controls, in robotics, medical devices, and also in some avionics systems.

Controller Area Network is a broadcast digital bus designed to operate at speeds from 20 kbit/s to 1 Mbit/s, standardized as ISO/DIS 11898 [6] for high speed applications (500 kbit/s) and ISO 11519-2 [7] for lower speed applications (up to 125 kbit/s). The transmission rate depends on the bus length and transceiver speed. CAN is an attractive solution for embedded control systems because of its low cost, light protocol management, the deterministic resolution of the contention, and the built-in features for error detection and retransmission. Controllers supporting the CAN communication standard, as well as sensors and actuators that are manufactured for communicating data over CAN, are today widely available. CAN networks are successfully replacing point-to-point connections in many application domains.

Commercial and open source implementation of CAN drivers and middleware software is today available from several sources, and support for CAN is included in automotive standards, including OSEKCom and AUTOSAR. The standard has been developed with the objective of time determinism and support for reliable communication. With respect to these properties, it has been widely studied by academia and industry, and methods and tools have been developed for predicting the time and reliability characteristics of messages.

The CAN standard has originally been proposed for application to automotive systems, but with time emerged as a quite appropriate solution for other control systems as well. This book tries a general approach to the subject, but in many places it refers to automotive standards and systems. Automotive architectures and functions are also often used as examples and benchmarks. We hope this is not a

problem for the reader and hopefully, the generalization of the proposed approaches and methods to other domains should not be too hard. On the other side, automotive systems are today an extremely interesting context for whoever is interested in complex and distributed embedded systems (including of course those using the CAN bus). They offer complex architectures with multiple buses and nodes with gateways, real time and reliability constraints and all the challenges that come with a high bus utilization and the demand for increased functional complexity.

This book attempts at providing an encompassing view on the study and use of the CAN bus, with references to theory and analysis methods, and a description of the issues in the practical implementation of the communication stack for CAN and the implications of design choices at all levels, from the selection of the controller, to SW development and architecture design. We believe such an approach may be of advantage to those interested in the use of CAN, from students of embedded system courses, to researchers, architecture designers, system developers, and all practitioners that are interested in the deployment and use of a CAN network and its nodes.

As such, the book attempts at covering all aspects of the design and analysis of a CAN communication system. Chapter 1 contains a short summary of the standard, with emphasis on the bus access protocol and on the protocol features that are related to or affect the reliability of the communication. Chapter 2 describes the functionality, the design, the implementation options, and the management policies of the hardware controllers and software layers in CAN communication architectures. Chapter 3 focuses on the worst case time analysis of the message response times or latencies. Chapters 4 and 5 presents the stochastic and statistical timing analyses. Chapter 6 addresses reliability issues. Chapter 7 deals with the analysis of message traces. Chapter 8 describes commercial tools for configuring, analyzing and calibrating a CAN communication system. Chapter 9 contains a summary of the main transport level and application-level protocols that are based on CAN.

Pisa, Italy
Montreal, QC, Canada
Palo Alto, CA, USA
Berkeley, CA, USA

Marco Di Natale
Haibo Zeng
Paolo Giusto
Arkadeb Ghosal

Contents

1	The CAN 2.0b Standard	1
1.1	Physical Layer	2
1.1.1	Bit Timing	2
1.1.2	Transceivers, Network Topology and Bus Length	6
1.1.3	Physical Encoding of Dominant and Recessive States	10
1.1.4	Connectors	10
1.1.5	The Physical Layer in ISO and SAE Standards	11
1.2	Message Frame Formats	13
1.2.1	DATA Frame	14
1.2.2	Remote Frame	17
1.2.3	Error Frame	17
1.2.4	Overload Frame	17
1.3	Bus Arbitration	17
1.4	Message Reception and Filtering	19
1.5	Error Management	20
1.5.1	CRC Checks	20
1.5.2	Acknowledgement	21
1.5.3	Error Types	21
1.5.4	Error Signalling	21
1.5.5	Fault Confinement	22
2	Reference Architecture of a CAN-Based System	25
2.1	CAN Controller and Bus Adapter	27
2.2	CAN Device Drivers	29
2.2.1	Transmission	29
2.2.2	Reception	31
2.2.3	Bus-Off and Sleep Modes	32
2.3	Interaction Layer	32
2.3.1	Direct Transmission Mode	35
2.3.2	Periodic Transmission Mode	36
2.3.3	Mixed Transmission Mode	37

2.3.4	Deadline Monitoring.....	38
2.3.5	Message Filtering.....	38
2.4	Implementation Issues.....	38
2.4.1	Driver Layer.....	39
2.4.2	Interaction Layer.....	41
3	Worst-Case Time Analysis of CAN Messages.....	43
3.1	Ideal Behavior and Worst-Case Response-Time Analysis.....	45
3.2	Analysis Flaw and Correction.....	48
3.3	Analysis of Message Systems With Offsets.....	50
3.4	Message Buffering Inside the Peripheral.....	51
3.5	An Ideal Implementation.....	52
3.6	Sources of Priority Inversion: When the Analysis Fails.....	57
3.6.1	Message Queue Not Sorted by Priority.....	57
3.6.2	Messages Sent by Index of the Corresponding TxObject.....	57
3.6.3	Impossibility of Preempting the TxObjects.....	58
3.6.4	Polling-Based Output.....	60
3.7	From the Timing Analysis of Messages to End-to-End Response-Times Analysis.....	62
3.8	Conclusions.....	65
4	Stochastic Analysis.....	67
4.1	Introduction.....	67
4.2	Reference Work.....	69
4.3	System Model and Notation.....	71
4.4	Stochastic Analysis of Message Response-Times.....	72
4.4.1	A Modeling Abstraction for CAN Messages.....	72
4.4.2	Stochastic Analysis of the Approximate System.....	77
4.5	Analysis on an Example Automotive System.....	84
5	Statistical Analysis.....	89
5.1	Introduction.....	89
5.2	Deriving and Fitting the Model.....	90
5.2.1	Common Characteristics of Message Response-Times.....	91
5.2.2	Fitting the Message Response-Times.....	93
5.3	Estimate of the Distribution Parameters.....	98
5.3.1	Parameters x^{off} and y	98
5.3.2	Parameters y^D and y^T	98
5.3.3	Parameters a and b of the Gamma Distribution.....	100
5.4	Predicting Message Response-Times.....	108
5.4.1	Prediction of Response-Time cdfs for Messages on the Reference Bus.....	110
5.4.2	Prediction of Response-Time cdfs for Messages on Other Buses.....	112
5.5	Comparison of Stochastic and Statistical Analyses.....	117
5.5.1	Input Information.....	117

5.5.2	Analysis Accuracy	118
5.5.3	Analysis Complexity	119
6	Reliability Analysis of CAN	121
6.1	Error Rates in CAN	122
6.2	Deviation Points in the Standard	124
6.2.1	Incorrect Values in the DLC Field	124
6.2.2	Dominant SRR Bit Value	125
6.3	Fault Confinement and Transition to Bus Off	125
6.4	Inconsistent Omissions or Duplicate Messages	126
6.5	Protocol Vulnerability	128
6.6	Bus Topology Issues	131
6.7	Babbling Idiot Faults and Bus Guardians	132
7	Analysis of CAN Message Traces	133
7.1	Notation	134
7.2	Case Studies	134
7.3	Trace Analysis	135
7.3.1	Identification of Reference Messages	136
7.3.2	Base Rate Message	137
7.3.3	Reconstruction of the True Message Periods and Arrival Times	137
7.3.4	Queuing Jitter Because of TxTask Scheduling Delays	139
7.3.5	Clock Drifts and Actual Message Periods	139
7.3.6	Finding the Message Phase with Respect to the Base Rate Message	141
7.4	Analysis of Automotive Case Studies	142
7.4.1	Overwrite Error at the Destination Node for Message 0x1F3 in V_1	144
7.4.2	Aperiodic Message Transmission	148
7.4.3	Local Priority Inversion	148
7.4.4	Remote Priority Inversion	152
7.4.5	Message Loss at Source Node	153
7.4.6	Reconstruction of Message-to-ECU Relation in a Hybrid Vehicle	155
8	CAN Tools	157
8.1	System Configuration	158
8.2	System Analysis	160
8.2.1	Network Bus Simulation and Rest-Bus Simulation	160
8.2.2	Traffic Logging and Analysis	167
8.2.3	Network Bus Worst/Best Case Analysis	169
8.3	Protocol Stack Implementation and Configuration	175
8.3.1	CAN Driver Development and Integration	175
8.4	Protocol Stack/Application Calibration	178

9 Higher-Level Protocols	181
9.1 J1939	181
9.1.1 Physical Layer	182
9.1.2 Parameters and Parameter Groups	183
9.1.3 Protocol Data Units (PDUs)	185
9.1.4 Transport Protocol	186
9.1.5 Network Management	188
9.2 CANopen	190
9.2.1 CANopen Architecture and Standards	190
9.2.2 Physical Level	190
9.2.3 Object Dictionary	192
9.2.4 Time Services	195
9.2.5 Communication Protocols	197
9.2.6 Service-Oriented Communication and SDO	197
9.2.7 Message Identifier Allocation	203
9.2.8 Network Management	204
9.3 CCP	206
9.3.1 Protocol Definition	206
9.3.2 Command Receive Object	207
9.3.3 Data Transmission Object	208
9.3.4 List of CCP Commands	208
9.3.5 List of Packet IDs and Command Return Codes	210
9.4 TTCAN	210
9.4.1 Motivation and Goal	211
9.4.2 Synchronization Mechanisms in TTCAN	211
9.4.3 Scheduling of TTCAN Networks	212
9.4.4 Reliability of TTCAN and Additional Notes	214
Symbols	215
References	217
Index	221

List of Figures

Fig. 1.1	Bit propagation delay	4
Fig. 1.2	Definition of the bit time and synchronization	4
Fig. 1.3	The NXP PCA 82C250 transceiver and its connections	6
Fig. 1.4	Relationship between bus length and bit rate for some possible configurations	7
Fig. 1.5	Terminating a high-speed network	9
Fig. 1.6	Placement of termination resistors on a low-speed network	9
Fig. 1.7	Encoding of dominant and recessive bits	10
Fig. 1.8	The bus connector defined by CIA (DS 102-1)	10
Fig. 1.9	The CAN data frame format	14
Fig. 1.10	The CAN identifier format	15
Fig. 1.11	The control field format	16
Fig. 1.12	The CRC and acknowledgement formats	16
Fig. 1.13	An example of CAN bus arbitration based on message identifiers .	19
Fig. 1.14	Masks and filters for message reception	20
Fig. 1.15	Bit sequence after detection of an error	22
Fig. 1.16	Node error states	23
Fig. 2.1	Typical architecture of a CAN-based system. Actual systems may have different dependencies among layers	26
Fig. 2.2	Interaction layer architecture	33
Fig. 2.3	Timing for the transmission of direct mode message objects	35
Fig. 2.4	Timing for the transmission of periodic mode messages objects ...	36
Fig. 2.5	Timing for the transmission of mixed mode messages objects	37
Fig. 2.6	Interrupt- or polling-based management of the TxObject at the CAN peripheral	40
Fig. 2.7	The middleware task TxTask executes with period T_{TX} , reads signal variables are enqueues messages	42

Fig. 3.1	Worst-case stuffing sequence, one bit is added for every four (except for the first five) bit in the frame	46
Fig. 3.2	Worst-case response-time, busy period and time critical instant for m_i	47
Fig. 3.3	An example showing the need for a fix in the evaluation of the worst-case response-time	49
Fig. 3.4	In systems with offsets the analysis must be repeated for all the message queuing times inside the system hyperperiod ..	50
Fig. 3.5	Static allocation of TxObjects	52
Fig. 3.6	Temporary queuing of outgoing messages	53
Fig. 3.7	Buffer state and possible priority inversion for the single-buffer (<i>top</i>) and two-buffer (<i>middle</i>) cases. Priority inversion can be avoided with three-buffers (<i>bottom</i>)	54
Fig. 3.8	Priority inversion for the single buffer case	55
Fig. 3.9	Priority inversion for the two buffer case	56
Fig. 3.10	Priority inversion when the TxObject cannot be revoked	58
Fig. 3.11	Double software queue	59
Fig. 3.12	Trace of a priority inversion for a double software queue	59
Fig. 3.13	The transmit polling task introduces delays between transmission ..	60
Fig. 3.14	Priority inversion for polling-based output	61
Fig. 3.15	Latency distribution for a message with polling-based output compared to interrupt-driven output	61
Fig. 3.16	Model of a distributed computation and its end-to-end latency	64
Fig. 3.17	Data loss and duplication during communication	64
Fig. 4.1	Convolution and shrinking	70
Fig. 4.2	An example of characteristic message transmission time	73
Fig. 4.3	The transformation of the queuing model for remote messages	74
Fig. 4.4	Updating the backlog in the discrete-time model	78
Fig. 4.5	The response-time cdfs of high priority messages m_5 and m_{25} in Table 4.1	86
Fig. 4.6	The response-time cdfs of low priority messages m_{43} and m_{63} in Table 4.1	87
Fig. 4.7	The response-time cdf of a low priority message in a bus trace	88
Fig. 5.1	Response-time cdf of m_{25} with more than one higher priority harmonic set	92
Fig. 5.2	Fitting a mix model distributions to the response-time of m_{13}	95
Fig. 5.3	Quantile–quantile plot of samples from simulation and fitted distribution for m_5	96
Fig. 5.4	Quantile–quantile plot of samples from simulation and fitted distribution for m_{69}	97
Fig. 5.5	Absolute errors in the estimation of the y^D values for messages ...	99
Fig. 5.6	Approximate linear relation between μ and Q	102
Fig. 5.7	Approximate linear relation between b and Q	103

Fig. 5.8	Approximate linear relation between μ and Q^{hr}	104
Fig. 5.9	Approximate linear relation between b and Q^{hr}	104
Fig. 5.10	Minimized absolute errors in the estimation of μ and b for messages on the reference bus	106
Fig. 5.11	Minimized relative errors in the estimation of μ and b for messages on the reference bus	110
Fig. 5.12	Prediction of the response-time cdf for message m_5	111
Fig. 5.13	Prediction of the response-time cdf for message m_{68}	111
Fig. 5.14	Prediction of response-time cdf for m_{39} with more than one harmonic set	112
Fig. 5.15	Prediction of response-time cdf for messages on bus2: worst result	114
Fig. 5.16	Prediction of response-time cdfs for messages on bus2: better quality result	115
Fig. 5.17	RMSE of statistical analysis: reference bus, bus2, and bus3	116
Fig. 5.18	RMSE of statistical analysis: message traces	117
Fig. 5.19	Comparison of stochastic and statistical analyses: RMSE	118
Fig. 5.20	Comparison of stochastic and statistical analyses: K-S statistics ..	119
Fig. 6.1	The experimental setup for measuring bit error rates in [26]	122
Fig. 6.2	A scenario leading to inconsistent message duplicates	127
Fig. 6.3	Two bit errors result in a much longer sequence because of stuffing	129
Fig. 6.4	Probability of a corrupted message being undetected for a given number of bit errors	130
Fig. 7.1	Signal overwrite error	135
Fig. 7.2	From message arrival to message receive	136
Fig. 7.3	Detection of idle time on the CAN bus	137
Fig. 7.4	Defining a reference periodic base for the arrival times	138
Fig. 7.5	Calculation of period for message 0x128 on an experimental vehicle considering clock drift	140
Fig. 7.6	Detection of the message phase	141
Fig. 7.7	Response time of messages 0x184 and 0x2F9 on E_{12} of vehicle V_1	145
Fig. 7.8	Response time of messages 0x12A , 0x1F3 and 0x3C9 on E_{11} of vehicle V_1	146
Fig. 7.9	An aperiodic message 0x130 from E_{22} on vehicle V_2 : activation interval in general is 200 ms, with additional activations every 1,500 ms indicated by the triangles	149
Fig. 7.10	An aperiodic message 0x300 from E_{23} on vehicle V_2 : two sets of periodic transmissions at nominal period 100 ms indicated by <i>triangles</i> and <i>arrows</i> respectively, with relative phase = 80 ms/20 ms	149

Fig. 7.11	Local priority inversion of message 0x124 on node E_{23} of vehicle V_2 , as shown in the trace segment. Note that on node E_{23} the period of TxTask should be 10 ms, the <i>gcd</i> of the periods. 0x124 should always be queued together with 0x170 , 0x308 , 0x348 , 0x410 and 0x510 , but it is transmitted after 0x170 , 0x308 , 0x348 as indicated by the <i>triangles</i>	150
Fig. 7.12	Local priority inversion of message 0x199 on node E_{15} of vehicle V_1 , as shown in the trace segment. Note that on node E_{15} the period of TxTask should be 12.5 ms, the <i>gcd</i> of the periods. 0x199 should always be queued together with 0x4C9 , but it is transmitted after 0x4C9 as indicated by the <i>triangles</i>	150
Fig. 7.13	Local priority inversion of message 0x19D on node E_{15} of vehicle V_1 , as shown in the trace segment. 0x19D should always be queued together with 0x4C9 and 0x77F , but it is transmitted after 0x4C9 or 0x77F as indicated by the <i>triangles</i>	151
Fig. 7.14	Local priority inversion of message 0x1F5 on node E_{15} of vehicle V_1 , as shown in the trace segment. 0x1F5 should always be queued together with 0x4C9 and 0x77F , but it is transmitted after 0x4C9 or 0x77F as indicated by the <i>triangles</i>	151
Fig. 7.15	Interpretation of possible cause of local priority inversion for message 0x124	152
Fig. 7.16	Remote priority inversion of message 0x151 on E_{24} of vehicle V_2 . 0x151 should always be queued together with 0x150 , but 0x380 , 0x388 , and 0x410 get transmitted between 0x150 and 0x151 , as indicated by the <i>triangles</i>	153
Fig. 7.17	Trace segment of message loss at the transmission node: 0x199 from E_{15} on vehicle V_1	153
Fig. 7.18	Trace segment of message loss at the transmission node: 0x19D from E_{15} on vehicle V_1	154
Fig. 7.19	Clock drifts for ECUs on the hybrid vehicle	155
Fig. 8.1	The CAN communication system tool flow	158
Fig. 8.2	The signal specification in CANdb++	159
Fig. 8.3	Rest-bus simulation in CANoe (source: Vector Informatik)	161
Fig. 8.4	Modeling and simulation of a CAN network in VisualSim (source: Mirabilis Design)	162
Fig. 8.5	VisualSim histogram of bytes transmitted over CAN Bus (source: Mirabilis Design)	164
Fig. 8.6	VisualSim plot of the CAN bus states (source: Mirabilis Design) ..	164

Fig. 8.7	VisualSim plot of CAN bus message latencies (source: Mirabilis Design)	165
Fig. 8.8	chronVAL GUI (source: INCHRON)	166
Fig. 8.9	chronVAL Gantt chart (source: INCHRON)	166
Fig. 8.10	chronBUS GUI (source: INCHRON)	168
Fig. 8.11	chronBUS Gantt chart (source: INCHRON)	168
Fig. 8.12	CANalyzer configuration and analysis (source: Vector Informatik)	169
Fig. 8.13	Traffic data log (source: Vector Informatik)	170
Fig. 8.14	SymTA/S bus timing model	172
Fig. 8.15	SymTA/S timing analysis and forecast results	172
Fig. 8.16	chronVAL Gantt chart (source: INCHRON)	173
Fig. 8.17	chronBUS table of results (source: INCHRON)	174
Fig. 8.18	chronBUS single message results (source: INCHRON)	174
Fig. 8.19	chronBUS event spectrum viewer (source: INCHRON)	175
Fig. 8.20	η^+ -CANbedded basic software	176
Fig. 8.21	η^+ -CANbedded basic software abstraction layers	176
Fig. 8.22	Vector configuration tools for CAN	177
Fig. 8.23	GENy configuration tool	178
Fig. 9.1	J1939 standards in the ISO/OSI reference model	182
Fig. 9.2	J1939 PDU definition inside the CAN identifier	185
Fig. 9.3	Message sequence for a multi-packet message that is of type broadcast (<i>left side</i>) as well as point-to-point (<i>right side</i>) in J1939	186
Fig. 9.4	J1939 address claiming procedure starting with a Request for Address Claim	189
Fig. 9.5	The CANopen standards, from the application to the physical layer	191
Fig. 9.6	The CANopen pin connections for a 5-pin mini connector	192
Fig. 9.7	The CANopen Object table index	193
Fig. 9.8	The CANopen Object table indexes for object types	193
Fig. 9.9	The CANopen Object table indexes for the Communication Profile Objects	194
Fig. 9.10	The error object register bits and their meaning	194
Fig. 9.11	The CANopen Identity Object structure with the bit map for the fields Vendor ID and Revision number	195
Fig. 9.12	Synchronization message, with the corresponding period and synchronization window	196
Fig. 9.13	The time stamp object	197
Fig. 9.14	The transfer protocol for SDO downloads and uploads	198
Fig. 9.15	SDO Transfer Initiate message and its reply from the server	199
Fig. 9.16	The initiate transfer message indicates the object dictionary entry on which to operate	199
Fig. 9.17	The SDO format for segmented transfers	200
Fig. 9.18	SDO download block exchange	200

Fig. 9.19	Message exchanges for Read- or Write-type PDOs	201
Fig. 9.20	Mapping of application objects into PDOs	201
Fig. 9.21	Synchronous and asynchronous PDO messages.....	202
Fig. 9.22	A taxonomy of PDO transmission modes	202
Fig. 9.23	Cyclic and acyclic synchronous PDOs.....	203
Fig. 9.24	The CANopen Object table indexes for object types.....	204
Fig. 9.25	The device states controlled by the Network Management protocol.....	205
Fig. 9.26	The three sub-states in initialization state.....	206
Fig. 9.27	Communication flow between CCP master and slave devices	207
Fig. 9.28	Message format of a Command Receive Object.....	208
Fig. 9.29	Format of a Command Return Message or an Event Message.....	208
Fig. 9.30	Format of a Data Acquisition Message	208
Fig. 9.31	An example of a TTCAN system matrix.....	213

List of Tables

Table 1.1	Typical transmission speeds and corresponding bus lengths	7
Table 1.2	Bus cable characteristics	8
Table 1.3	Acceptable voltage ranges for CAN transmitters and receivers ...	12
Table 1.4	The rules of updating transmit error count and receive error count	23
Table 2.1	CAN controllers	28
Table 3.1	Summary of properties for some existing CAN controllers	51
Table 4.1	An automotive CAN system with 6 ECUs and 69 messages	85
Table 5.1	An example automotive CAN system	91
Table 5.2	Statistics of the fitted model distributions for messages on the reference bus	97
Table 5.3	Coefficients $\beta_1 - \beta_4$ of the parameterized model of y^D for the reference bus	101
Table 5.4	Coefficients $\beta_5 - \beta_{19}$ of the parameterized model to minimize absolute errors of μ and b	107
Table 5.5	Coefficients $\beta'_5 - \beta'_{21}$ of the parameterized model to minimize relative errors of μ and b	109
Table 5.6	Error of the predicted distribution for messages on the reference bus with regression functions (5.9), (5.13), and (5.14)	113
Table 5.7	Error of the predicted distribution for messages on the reference bus with regression functions (5.9), (5.18), and (5.19)	114
Table 5.8	Comparison of stochastic and statistical analyses: analysis complexity	119

Table 6.1	Bit error rates on a CAN bus measured in three different environments	123
Table 6.2	Probability of inconsistent message omissions and inconsistent message duplicates	128
Table 6.3	Number of undetected message errors per day as a function of the bit error rate.....	131
Table 7.1	True periods estimated for the messages of node E_{12}	139
Table 7.2	Evidence of clock drift on the same node: message 0x128 on an experimental vehicle with different number of frames received in the same length of time	140
Table 7.3	Nodes and messages from a subset of vehicle V_1	143
Table 7.4	Nodes and messages from a subset of vehicle V_2	144
Table 7.5	Trace segment with arrival times and response times	147
Table 7.6	Messages from the hybrid vehicle (periods in <i>ms</i>)	154
Table 8.1	Characteristic functions of most relevant event streams.....	171
Table 9.1	Physical layer requirements for J1939	183
Table 9.2	The definition of engine temperature parameter group	184
Table 9.3	Parameters required for the definition of the connection management PGN and message format	187
Table 9.4	Definition of an ECU name in J1939	188
Table 9.5	Acceptable bit rates, bus lengths, and bit timings as specified by CANopen	191
Table 9.6	Pin assignment for the 5-pin mini connector.....	192
Table 9.7	Definition of the CAN identifier based on the function code and the device identifier	204
Table 9.8	List of CCP commands.....	209
Table 9.9	List of packet IDs and their meaning	210
Table 9.10	List of command return codes	210

Chapter 1

The CAN 2.0b Standard

This chapter introduces version 2.0b of the CAN Standard. This introduction is an excerpt of the main features of the protocol as described in the official Bosch specification document [16]. For more details, the reader should check the free official specification document available on-line, along with the other references provided throughout this chapter.

The CAN network protocol has been defined to provide deterministic communication in complex distributed systems with the following features and capabilities:

- Message priority assignment and guaranteed maximum latencies.
- Multicast communication with bit-oriented synchronization.
- System-wide data consistency.
- Bus multimaster access.
- Error detection and signaling with automatic retransmission of corrupted messages.
- Detection of permanent failures in nodes, and automatic switch-off to isolate faulty node.

In the context of the ISO/OSI reference model, the original CAN specification, developed by Robert Bosch GmbH, covers only the *Physical* and *Data link* layers. Later, ISO provided its own specification of the CAN protocol, with additional details for the implementation of the physical layer.

Generally speaking, the purpose of the Physical Layer is to define the encoding of bits into (electrical or electromagnetic) signals with defined physical characteristics. The signals are transmitted over wired or wireless links from one node to another. In the Bosch CAN standard, however, the description is limited to the definition of the bit timing, bit encoding, and synchronization. The specification of the physical transmission medium including the required (current/voltage) signal levels, the connectors, and other characteristics that are necessary for the definition of the driver/receiver stages and the physical wiring is not covered. Other reference documents and implementations have filled this gap, providing solutions for the practical implementation of the protocol.

The Data-link layer consists of the Logical Link Control (LLC) and Medium Access Control (MAC) sublayers. The LLC sublayer provides all the services for the transmission of a stream of bits from a source to a destination. In particular, it defines:

- Services for data transfer and remote data request.
- Conditions upon which received messages should be accepted, including message filtering.
- Mechanisms for recovery management and flow management (overload notification).

The MAC sublayer is considered as the kernel of the CAN protocol. The MAC sublayer is responsible for message framing, arbitration of the communication medium, acknowledgment management, and error detection and signaling. For the purpose of fault containment and additional reliability, the MAC operations are supervised by a controller entity monitoring the error status and limiting the operations of a node if a possible permanent failure is detected.

The following sections provide more details into each sub-layer, including requirements and operations.

1.1 Physical Layer

As stated in the introduction, the Bosch CAN standard defines bit encoding, timing and synchronization, included in the Physical Signaling (PS) portion of the ISO-OSI physical layer. The standard does not cover other issues related to the physical layer, including the types of cables and connectors that should be used for communicating over a CAN network and the ranges of voltages and currents that are considered as acceptable for the operations. In the OSI terminology, the Physical Medium Attachment (PMA) and Medium Dependent Interface (MDI) are the two parts of the physical layer which are not defined by the original standard.

1.1.1 Bit Timing

The signal type is digital with *Non Return to Zero* (NRZ) bit encoding. The use of NRZ encoding ensures a minimum number of transitions and high resilience to external disturbance. The two bits are encoded in physical medium states defined as “recessive” and “dominant.” (0 is typically assumed to be associated with the “dominant” state). The protocol allows multi-master access to the bus. At the lowest level, if multiple masters try to drive the bus state at the same time, the “dominant” configuration always prevails upon the “recessive.”

Nodes are requested to be synchronized on the bit edges so that every node agrees on the value of the bit currently transmitted on the bus. To achieve synchronization,

each node implements a protocol that keeps the receiver bit rate aligned with the actual rate of the transmitted bits. The synchronization protocol uses transition edges to resynchronize nodes. Hence, long sequences without bit transitions should be avoided to ensure limited drift among the node bit clocks. This is the reason why the protocol employs the so-called “bit stuffing” or “bit padding” technique, which forces a complemented bit in a transmission sequence after 5 bits of the same type. Stuffing bits are automatically inserted by the transmission node and removed at the receiving side before processing the frame contents.

Synchronous bit transmission enables the CAN arbitration protocol and simplifies data-flow management, but also requires a sophisticated synchronization protocol. Bit synchronization is performed first upon the reception of the start bit available with each asynchronous transmission. Later, to enable the receiver(s) to correctly read the message content, continuous resynchronization is required. Other features of the protocol influence the definition of the bit timing. Bus arbitration, message acknowledgement and error signalling are based on the capability of the nodes to change the status of a transmitted bit from recessive to dominant. Since the bus is shared, all other nodes in the network are informed of the change in the bit status before the bit transmission ends. Therefore, the bit (transmission) time must be at least large enough to accommodate the signal propagation from any sender to any receiver and back to the sender.

The bit time needs to account for a propagation delay that includes the signal propagation delay on the bus as well as delays caused by the electronic circuitry of the transmitting and receiving nodes. In practice, this means that the signal propagation is determined by the two nodes within the system that are farthest apart from each other as the bit is broadcasted to all nodes in the system (Fig. 1.1).

The leading bit edge from the transmitting node (node A in Fig. 1.1 reaches node B after the signal propagates all the way from the two nodes. At this point, B can change its value from recessive to dominant, but the new value will not reach A until the transition from recessive to dominant propagates across the entire bus length from B back to A. Only then can node A safely determine whether the signal level it wrote on the bus is the actual stable level for the bus at the bit sampling time, or whether it has been replaced (in case it was recessive) by a dominant level superimposed by another node.

Considering the synchronization protocol and the need that all nodes agree on the bit value, the nominal bit time (reciprocal of the bit rate or bus speed) is defined as composed of four segments (Fig. 1.2, segment names are all upper case in accordance with the notation in the original specification.)

- *Synchronization segment (SYNC_SEG)* This is a reference interval, used for synchronization purposes. The leading edge of a bit is expected to lie within this segment.
- *Propagation segment (PROP_SEG)* This part of the bit time is used to compensate for the (physical) propagation delays within the network. It is twice the sum of the signal propagation time on the bus line, plus the input comparator delay, and the output driver delay.

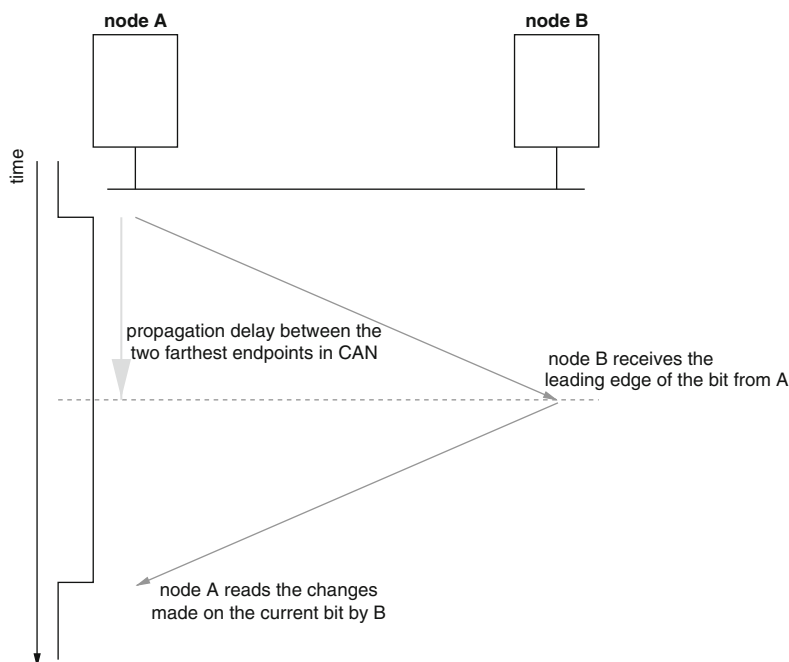


Fig. 1.1 Bit propagation delay

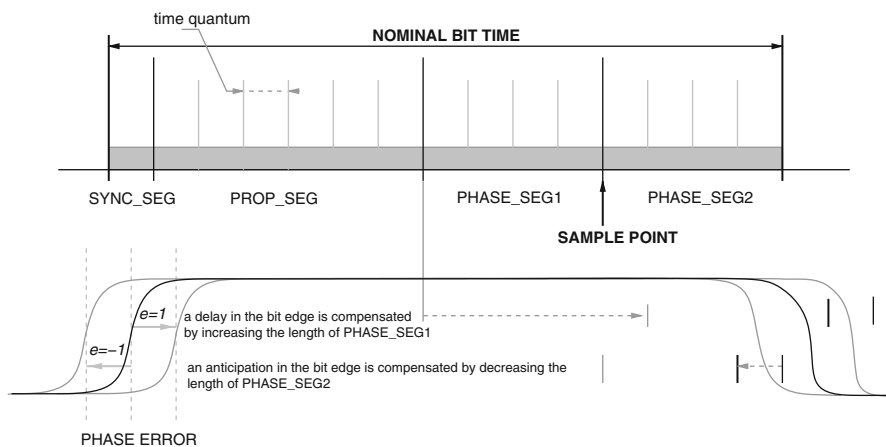


Fig. 1.2 Definition of the bit time and synchronization

- *Phase segments (PHASE_SEG1 and PHASE_SEG2)* These phase segments are time buffers used to compensate for phase errors in the position of the bit edge. These segments can be lengthened or shortened to resynchronize the position of SYNC_SEG with respect to the following bit edge.
- *Sample point (SAMPLE_POINT)* The sample point is the point of time at which the bus level is read and interpreted as the value of that respective bit. The quantity *information processing time* is defined as the time required to convert the electrical state of the bus, as read at the SAMPLE_POINT into the corresponding bit value.

All these segments are multiple of a predefined time unit, denoted as *time quantum* and derived from the local oscillator by applying a prescaler to a clock with rate *minimum time quantum*.

$$\text{time quantum} = m \times \text{minimum time quantum} \quad (1.1)$$

where m is the value of the prescaler. The *time quantum* is the minimum resolution in the definition of the bit time and the maximum assumed error for the bit-oriented synchronization protocol. The widths of the segments are defined by the standard as follows: SYNC_SEG is equal to 1 *time quantum*; PROP_SEG and PHASE_SEG are between 1 and 8 times the *time quantum*; PHASE_SEG2 is the maximum between PHASE_SEG1 and the *information processing time*, which must be always less than or equal to twice the *time quantum*.

Two types of synchronization are defined: hard synchronization, and re-synchronization (Fig. 1.2).

- *Hard synchronization* takes place at the beginning of the frame, when the start of frame bit (see the frame definition in the following section) changes the state of the bus from recessive to dominant. Upon detection of this edge, the bit time is resynchronized in such a way that the end of the current quantum becomes the end of the synchronization segment SYNC_SEG. Therefore, the edge of the start bit lies within the synchronization segment of the restarted bit time.
- *Re-synchronization* takes place during transmission. The phase segments are shortened or lengthened so that the following bit starts within the SYNC_SEG portion of the following bit time. In detail, PHASE_SEG1 may be lengthened or PHASE_SEG2 may be shortened. Damping is applied to the synchronization protocol. The amount of lengthening or shortening of the PHASE_SEG segments is upper bounded by a programmable parameter *resynchronization jump width*, which is set to be between 1 and $\min(4, \text{PHASE_SEG1})$ times the *time quantum*.

Nodes can perform synchronization only when the bus state changes because of a bit value change. Therefore, the possibility of resynchronizing a bus unit during a frame transmission depends on the possibility of detecting at least one bit value transition in any interval of a given length. The bit stuffing protocol guarantees that the time interval between any two bit transitions is upper bounded (no more than 5 bits) in such a way that clocks should never drift beyond the possibility of recovery offered by the synchronization protocol.