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

Springer

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

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 — Marco Di Natale
Montreal, QC, Canada — Haibo Zeng
Palo Alto, CA, USA — Paolo Giusto
Berkeley, CA, USA — Arkadeb Ghosal

Contents

List of Figures

List of Tables

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.

M. Di Natale et al., *Understanding and Using the Controller Area Network Communication Protocol: Theory and Practice*, DOI 10.1007/978-1-4614-0314-2_1,

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.

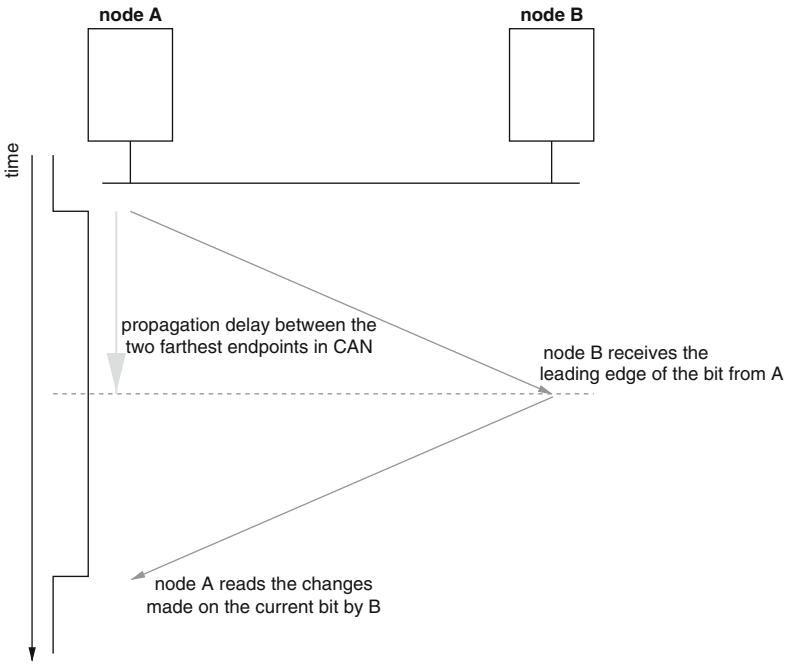

Fig. 1.1 Bit propagation delay

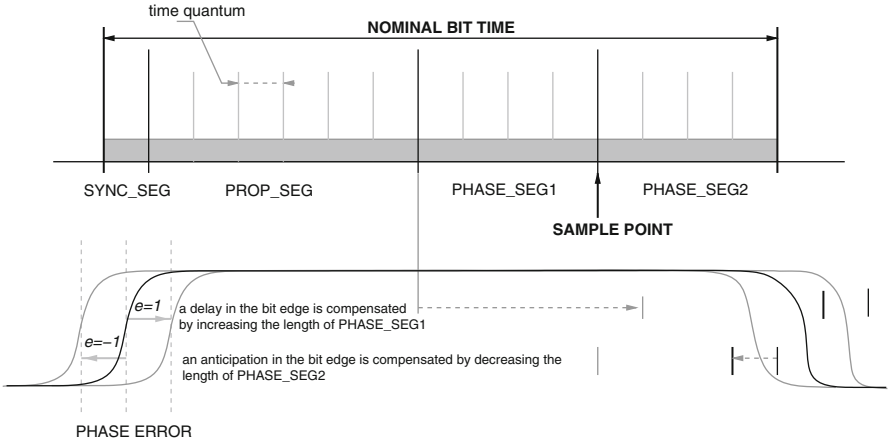

Fig. 1.2 Definition of the bit time and synchronization

- *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 SYNCH_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*.

$$\text{time quantum} = m \times \text{minimum time quantum} \tag{1.1}$$

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 SYNCH_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, 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.