Electronics Library

Editor: Prof. Dr. Ing. Andreas Grzemba

MOST®

THE AUTOMOTIVE
MULTIMEDIA NETWORK

from MOST25 to MOST150

FRANZIS

Electronics Library

MOST

Electronics Library

Editor: Prof. Dr. Ing. Andreas Grzemba

MOST®

THE AUTOMOTIVE
MULTIMEDIA NETWORK

FROM MOST25 TO MOST150

FRANZIS

The MOST System

This book is based on the MOST Specification Version 3.0 E2, 07/2010. Since the specification is under constant development there could have been changes from what is described in the book. Should there be any differences between the book and the specification, the specification should be followed. The latest version of the specification can be found at www.mostcooperation.com.

Typesetting: Machelett Software & Consulting, Luise-Riegger-Str. 30, D-76137 Karlsruhe
art & design: www.ideehoch2.de
Printing: Bercker, 47623 Kevelaer
Printed in Germany

ISBN 978-3-645-**65061-8**

Preface to the second edition

The first edition of this book was released in March 2008 and based on the first generation MOST25 (MOST Specification 2.4). It operates at 25 Mbps. Now, the second edition covers additionally the second and the third generation. MOST is the abbreviation for *Media Oriented Systems Transport.*

With the MOST Specification Rev. 2.5 for MOST50 – the second generation – the MOST Cooperation doubles the bandwidth for automotive infotainment solutions from 25 to 50 Mbps. The MOST Specification of Electrical Physical Layer Rev. 1.1 is an additional key specification that enables data transmission over an unshielded twisted pair (UTP) of copper wires while meeting the stringent automotive electromagnetic compatibility requirements.

MOST150 is the third generation offering a bandwidth, of 150 Mbps. With the transmission over legacy POF/LED optical physical layer MOST150 offers a smooth migration from MOST25 and MOST50 allowing carmakers to continue to use POF and LEDs as light sources. In addition to higher bandwidth MOST150 features an isochronous transport mechanism to support extensive video applications, as well as an Ethernet channel for efficient transport of IP-based packet data. This channel carries legacy Ethernet packets (according to IEEE 802.3) without change. Thus, the new generation of MOST provides the automotive-ready physical layer for Ethernet in the car. This way, MOST will be open to a broad variety of IP based applications. It even allows to adjust the bandwidth of conventional streaming connection on the one hand and the IP communication on the other according to the corresponding requirements. Besides, MOST150 still supports the well-known asynchronous channel to ensure backward compatibility of MOST25 applications. With MOST150, audio and video signals can be transported with high bandwidth efficiency and without any overhead for addressing, collision detection/ recovery or broadcast. Multiple high definition (HD) video streams and multi-channel surround sound with premium quality of service can be transmitted, while simultaneously moving high loads of packet data around.

MOST network needs to support digital audio, video and data services equally:

- High QoS Synchronous (PCM audio …)
- High QoS Isochronous (MPEG Transport Streams, audio and video over IP …)
- High performance Packet with IP support
- Realtime Control

The number of vehicle models relying on the MOST infotainment backbone has now hit the one hundred mark. This fast implementation growth rate with a total of 100 vehicle models on the road today reflects the practical suitability and reliability of MOST. Only three years after the MOST Cooperation was founded in 1998, the first MOST car was introduced in 2001. The following year, 13 more models implemented the MOST infotainment backbone. In October 2007, the leading network technology had found its way into 50 car models, climbing to 68 models in March, 2009. Just one year later there are 50% more models on the road as MOST Technology spreads into the mass market.

Asian carmakers presented their first vehicles with MOST inside in 2007. Today, there are 22 models manufactured by Asian automakers with MOST built in. MOST Technology is now heading to the American continent with the first US carmaker working on future vehicles that incorporate the MOST network.

MOST25 Technology has established itself in the European and Korean markets. The Japanese and US markets prefer MOST50.

Now the newest Specification Rev. 3.0 for MOST150 is on the way to production. Audi and Daimler will be the first carmakers to integrate MOST150 into series production vehicles.

The second edition has a new structure. Whereas the first edition, MOST also ranks to the in-car communication systems, the second edition focuses solely on MOST.

Acknowledgements

I would like to thank my co-authors for spending their precious time to cooperate on this book and thus make it possible at all and to all members of the MOST Cooperation Steering Committee, who initiated and actively supported the second edition.

I also owe thanks to the coordinator of the MOST Cooperation, Mr. Wolfgang Bott for the comprehensive support and Mr. Renato Machelett for the complete proofreading of the script.

Finally, I would like to thank Franzis publishing house for their many ways of contributing to this book.

January 2011

Karlsruhe — Deggendorf

MOST Cooperation — Andreas Grzemba (Editor)

Preface to the first edition

The automotive industry can look back on nearly 20 years of experience using in-car communication systems. The development started with the CAN-Bus, which has been used in production vehicles since the 1990s. The infotainment sector in cars has developed rapidly within the last few years. Where an AM/FM tuner was initially sufficient, many vehicles are now equipped with high-grade sound and navigation systems. This also increases the number of devices involved – radio, cell phone, CD changer, navigation system, voice operation and an additional multi-channel amplifier supplement the primary systems. These devices must interact in concert with each other and can only be controlled via a central operating surface to ensure drivers are not unnecessarily distracted. The bandwidth of the CAN-Bus, which can only move control signals, but no audio and video signals, is no longer sufficient for these demands.

MOST, *Media Oriented Systems Transport*, a communication system with a new, flexible architecture, used by many different manufacturers, was developed to meet these demands. It can transmit audio signals synchronously, as in telephones, and is the most widely used multimedia system in cars today.

MOST technology is the result of the collaboration of car makers and suppliers, working to establish and refine a common standard within the MOST Cooperation. The founding fathers of the MOST Cooperation constitute the steering committee, the managing body. All companies interested in the system can, however, collaborate in working groups. Chapter 1 introduces the Cooperation.

The MOST system not only comprises all layers of the Open System Interconnect (OSI) Reference Model for communication and computer network design, but also standardizes the interfaces with the applications (e.g., an AM/FM tuner), which are defined as function blocks. Their implementation is described by example interaction models. In addition, the encoding of the audio and video signals is part of the specification. The system designer is thus able to design multi-media systems on a relatively high abstraction level.

This book is an introduction to the MOST technology. It cannot, however, replace the work with the specifications. The basis for this book is the MOST specification 2.4, taking into account further developments. The current specifications can be found at the MOST Cooperation website http://www.mostcooperation.com.

The book is divided into two parts.

Part 1 deals with the MOST standard. After introducing the structure of the MOST Cooperation, the communication architecture in the vehicle is presented for classifying the MOST Technology. Chapter 3, *System Architecture*, provides an introduction into the technology. The following chapters describe the most important aspects in detail.

Part 2 deals with the application of the MOST Technology. First, development tools are introduced. The next chapter explains the structure and implementation of a simple MOST system by means of these tools. This part then describes the basic structure of MOST gateways, the handling of MOST components and the use of MOST in current production vehicles. The last chapter describes the sound systems used in cars.

The **appendix** comprises a list of abbreviations, a glossary with the most important terms from the world of MOST, as well as a comparison between MOST, Ethernet and IEEE1394. A short survey of the other communication systems currently used in cars may contribute to a better understanding of the MOST technology. In the context of gateways, it is mainly the CAN-Bus and Bluetooth which are of interest.

I hope that this book will contribute to widely disseminating MOST Technology and that it will be a useful reference work for designers and users. MOST is a system which is subject to constant improvements and further developments. Ideas or comments concerning this book will be gratefully accepted at the e-mail address andreas.grzemba@fh-deggendorf.de.

Acknowledgements

I would like to thank my co-authors for spending their precious time to cooperate on this book and thus make it possible at all.

My further thanks go to all members of the MOST Cooperation Steering Committee, who initiated and actively supported the project.

I also owe thanks to Mr. Josef Berwanger of BMW for checking through chapter FlexRay, Mr. Roland Trißl of SMSC Europe for proof-reading chapter 9, Mr. Matthias Karcher of SMSC Europe, Dr. Carsten Kellner of Vector Informatik for cooperating on chapters 14 and 15, as well as Mr. Jochen Klaus-Wagenbrenner of Harman Automotive Division and Mr. Martin Arend of BMW for proof-reading chapter 16.

I would particularly like to thank Mr. Robert Reiter of BMW for his many ideas and suggestions, Dr. Christian Thiel, Mr. Armin Schmidt and Mr. Henry Muyshondt, and their colleagues of SMSC, as well as Dr. Alexander Leonhardi of Daimler for the complete proof-reading of the script.

I would also like to thank the translators for translating the book into English.

Finally, I would like to thank my wife Jana for the first thorough correction of the script and Mrs. Astrid Lehmann and her colleagues at Franzis publishing house for their many ways of contributing to this book.

Deggendorf, January 2008

Andreas Grzemba

(Editor)

The authors of the book in alphabetical order:

Dr.-Ing. Steffen Abbenseth VOLKSWAGEN AG	Chapter 14
Dipl.-Ing. (FH) Frank Bähren Harman Automotive Division	Chapter 9
Dipl.-Ing. (FH) Patrick Blum Harman Automotive Division	Chapter 13
Dr.-Ing. Wolfgang Bott Ingenieurbüro BOTTCO	Chapters 3, 12
Dr. Simon Burton, BEng. (Hons), Phd. Vector Informatik GmbH	Chapter 10
Dipl.-Ing. (FH) Markus Dittmann Harman Automotive Division	Chapter 16
Dr.-Ing. Volker Feil Daimler AG	Chapter 8
Prof. Dr.-Ing. Andreas Grzemba FH Deggendorf	Chapters 2, 5, 9, 11, 15, 17 Appendix B, C, D
Dr.-Ing. Dieter Jurzitza Herrmann Ultraschalltechnik GmbH	Chapter 16
Dipl.-Ing. (FH) Stefan Kerber SMSC Europe GmbH	Chapter 9
Dr.-Ing. Thomas Kibler Daimler AG	Chapter 6

Dipl.-Physiker Jochen Kirschbaum BMW AG	Chapter 7
Dr. Rainer König Daimler AG	Chapter 1
Dipl.-Ing. Joachim Leonhard K2L GmbH	Chapter 17
Dr. Alexander Leonhardi Daimler AG	Chapter 4
M.Sc. CS Markus Nordgren SMSC Sweden	Section 5.8.2 - 5.8.4
Dipl.-Ing. Tilo Nothacker Daimler AG	Chapter 6
Dipl.-Ing. Stefan Poferl Daimler AG	Chapter 6
Dipl.-Ing. (FH) Armin Schmidt SMSC Europe GmbH	Chapter 10
Dipl.-Ing. (FH) Alexander Schneidenbach AUDI AG	Chapter 14
Dr. M. Eng. Andreas Schramm BMW AG	Chapter 15 Appendix A
Dipl.-Ing. Harald Schöpp SMSC Europe GmbH	Chapter 18
Dr.-Ing. Christian Thiel SMSC Europe GmbH	Chapter 1

Contents

1 MOST Cooperation

1.1 Motivation and History

In the late 80s, when the first car networks for controlling devices and for carrying multiple signals over one conductor were implemented, all car makers were still of the opinion that such technologies had a high potential for differentiating their products from those of other companies. Mercedes developed the CAN system, while PSA (Peugeot Citroën) and Renault controlled their vehicle infrastructure via VAN. As this was not suitable for the USA, J1850 was developed there. BMW, too, had their own network, the I-Bus. It took some time for all parties to realize that these technologies all served the same purpose: to transmit bits and bytes between devices. There is little differentiation potential in that. The different technologies, however, meant a high expenditure of time and money for the car manufacturers. They all developed similar but different functions and transmission technologies. It took nearly twenty years until CAN became generally accepted. Nowadays, nearly all vehicles across the globe control their interior functions and power transmissions via various CAN networks. The efforts for developing the other networking technologies could have been spared, but in the 80s the time was not yet ripe for such conclusions.

In 1996, the beginnings of MOST were being discussed at BMW, in collaboration with Becker (today Harman Automotive Division) and OASIS Silicon Systems (today SMSC) – on the basis of the D2B system developed and put into production by Mercedes – and it was decided to carry on this development together with other vehicle manufacturers. To develop individual solutions would have meant for the companies to individually bear all expenses again. The first contacts to Daimler Benz were made, which were initially more of a cautious approach towards each other than intensive collaboration.

Through a careful process, which was positively carried on by both car makers, the parties became closer and it was decided to collaborate on the development of MOST. The initial phase was marked by the idea of only specifying and collaborating on network functions, such as the physical bus layer and the network management. The more intensive the collaboration and trust became, the easier it became to discuss the application requirements which arose for the bus system. This joint process, which the two car makers went through, was the foundation for the definition of the physical transmission layer and for the development of functional specifications. These specifications were later extended by compliance test specifications for the physical transmission layer, the network management (core compliance) and the compliance on the application level (see chapter 12).

This initial collaboration resulted in the following fundamental agreement, which characterizes the activities of the MOST Cooperation up to this day:

MOST was established to realize series developments. Instead of a theoretical standardization, there is collaboration on a practical basis, in order to specify and implement subjects which are required for use in series production. None of the parties involved will obstruct the others. The party which handles the most time-demanding requirements for the individual functional components sets the pace. Development requirements of those parties which cannot be realized within this time frame will be dealt with in the next release.

After the first specifications had been drawn up together and the first steps of a joint development had successfully been made, the idea came up to extend these activities. It was decided to found an open cooperation for the development and standardization of MOST. Other vehicle makers, device and component manufacturers could join this cooperation on their own account without having to take regulative hurdles.

In 1998 the MOST Cooperation was founded as a German civil law partnership (GbR) by BMW, Daimler Benz, Becker and OASIS Silicon Systems. Audi joined the founding group soon thereafter.

When the MOST Cooperation became better known, the number of members quickly increased to about 80. Nearly all vehicle manufactures around the world and nearly all makers of infotainment devices and many components manufacturers also joined.

The MOST Cooperation not only focuses on the technical development of MOST and its respective specifications, but also on the promotion of the technology to become a global standard for multimedia networks in vehicles. In order to achieve that, MOST Cooperation has engaged in commercial show events at exhibitions and congresses throughout the USA, Europe and Asia. The MOST Cooperation has been represented by its members on quite a number of exhibitions. At the ITS World Congress 2000 in Turin for instance, as a world premiere, several vehicle makers presented with MOST a jointly developed technology. The vehicle manufacturers involved at that time (Audi, BMW, Mercedes and Volkswagen) joined this event with vehicles and the responsible designers.

Today there is a lot of cooperation between vehicle manufacturers. It has become a matter of course that infrastructural technologies, which are invisible for the customer, are jointly developed.

Fig. 1.1: Exhibition booth of the MOST Cooperation at the ITS World Congress 2000 with vehicles from Audi, BMW, Mercedes and Volkswagen

1.2 Fast Standardization

When the MOST Cooperation was founded, only the Network Interface Controller OS8104 and a few specifications for the network management and the transport layer were at hand. MOST had to get ready for series production deadlines. A fast standardization was therefore required, which is normally a contradiction in itself. Standardization is usually associated with years of processing and endless discussions. How could a fast standardization be achieved?

In order to reach a broad standardization, a lot of companies have to contribute, but discussions must be efficient. Decisions must be made quickly and therefore in small groups. Only companies which are competent in a specific field should participate in the discussions. The operating mode of the Cooperation must be flexibly adaptable to the requirements. Apart from that, there should be no irrelevant topics on the agenda. The MOST Cooperation only works on projects which aim at becoming a standard in production vehicles. Each of the companies involved must accept and commit itself to the Cooperation’s philosophy (see section 1.1).

The specific structure of the MOST Cooperation was defined in order to bridge the contradiction between "fast" and standardization. It has proved reliable and has since served as an example for a number of other cooperations,

1.3 Structure and Specification Work

Figure 1.2 shows the organizational structure, which is explained in detail hereunder.

1.3.1 The Steering Committee

The MOST Cooperation consists of the group of founders, who make up the Steering Committee, and of Associated Members, who are not members of the Steering Committee. Only the members of the Steering Committee are partners of the MOST Cooperation. They have the duty to cooperate, whereas the associated members cooperate on a voluntary basis. All members, however, may use the results of the MOST Cooperation unrestrictedly and to the full extent. They grant each other free right to use their respective contribution to the standardization.

The cooperation contract of the MOST Cooperation is only a rough cooperation framework, which is brought to life by the Steering Committee. Within this framework, the Steering Committee defines how the standardization work should be done, i.e., as the legislative branch it defines the rules of cooperation and can flexibly adapt them to a specific situation. The rules are described in the *MOST Cooperation Organizational Procedures* [MOST Org] document.

The Steering Committee, moreover, reserves the formal right to sign off all MOST Cooperation specifications. It will use its right of veto only if absolutely necessary and, as a rule, follows the recommendations of the technical groups with respect to the specifications.

In addition to this, the Steering Committee takes care of the legal matters of the MOST Cooperation, as well as of the cooperation with other industrial forums and of the MOST Cooperation's promotion activities.

Apart from that, the Steering Committee is responsible for the MOST Cooperation budget. This includes, besides budget planning and control of the operating results, the statement of account for the members.

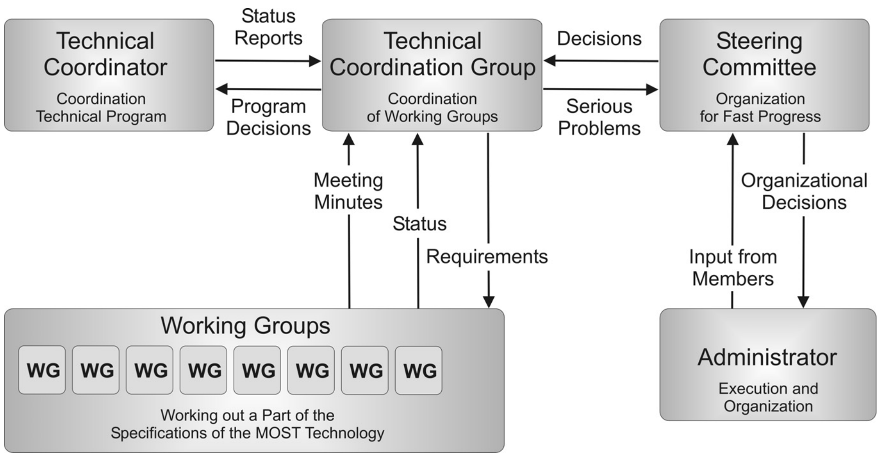

Technical Coordinator
Coordination Technical Program
Status Reports
Program Decisions
Technical Coordination Group
Coordination of Working Groups
Decisions
Serious Problems
Steering Committee
Organization for Fast Progress
Meeting Minutes
Status
Requirements
Organizational Decisions
Input from Members
Working Groups
WG
Working out a Part of the Specifications of the MOST Technology
Administrator
Execution and Organization

Fig. 1.2: Organization of the MOST Cooperation specification work

1.3.2 The Administrator

The Steering Committee appoints an administrator. He incorporates the decisions made by the Steering Committee into the daily business of the MOST Cooperation. He is also in charge of communication with the members or with applicants for membership. Additionally, he organizes the MOST Cooperation events and manages the website.

1.3.3 The Technical Coordination Group

The technical work of the MOST Cooperation is accomplished by the Technical Coordination Group (TCG), which is made up of the Steering Committee members and all other vehicle manufacturers. This ensures that the MOST Cooperation projects are implemented according to the requirements of the vehicle manufacturers and are not just any random standardizations.

The TCG defines projects, establishes Working Groups (WG) and determines who should be coordinator of a WG. This is mostly an employee of that member company of the TCG for which the results of the project are most important. This ensures that the WG is managed and urged on by the member with the highest interest in the project. The TCG also invites other member companies to cooperate in the new WG. This ensures that participants in a WG make useful contributions and that the WG can work in a target-oriented manner. The number of participants is thus also restricted to a reasonable number.

The TCG defines the aims of the WG and is regularly informed about the results. If necessary, the targets are adjusted to new requirements. Should there be insoluble conflicts about targets, the WG can inform the TCG which can then make further decision.

1.3.4 The Working Groups

A Working Group with a coordinator is put in place for each technical project of the MOST Cooperation. The WG deals with the project according to the objectives set up by the TCG. The organization of the work is up to the coordinator and the WG. The WG only has to comply with the frame conditions as defined by the *MOST Cooperation Organizational Procedures* [MOST Org]. This ensures that the individual groups can work according to their own philosophy and with their own means. This is necessary, as work on the physical bus layer for instance is subject to requirements different from work on the software level.

A WG usually creates one or several specifications. As soon as the WG has passed a specification, it is checked for conformity with the specification rules by the central WG of the MOST Cooperation - WG Device Architecture. If no corrections are necessary, the specification advances to the pre-release stage, where all members can inspect and comment on it. If there is no need for correction here either, the specification is released after eight weeks and becomes part of the official MOST Specifications.

1.3.5 The Technical Coordinator

The MOST Cooperation also has a Technical Coordinator. He is a project manager and controls the overall technical activities of the MOST Cooperation. He is the contact person for the WGs, keeps an overview of their status and reports it to the TCG. He coordinates processes, such as the release of specifications, which were worked on by different WGs. The Technical Coordinator’s main task over all is to ensure fast progress of the technical work.

Chapter 3 explains the structure of the MOST specifications.

1.4 The Compliance Verification Program

The MOST Cooperation aims at creating a standard, where all members profit from synergy effects, such as increased economies of scale for the key components. The members thus grant each other the free right to use their intellectual property rights (copyrights, trademark rights, patents, etc.) which are necessary for implementing the MOST specifications. All “owners” of MOST are interested in the specifications only being used for MOST purposes and not being misused in parts for other projects.

The process of Compliance Verification was defined in order to guarantee this aim. Each device, for which the manufacturer obtains a license under the protective rights of the members, must prove that it complies with the MOST specifications. For this purpose, the MOST Cooperation specified MOST Compliance Tests, which must be passed by each device. Only if a device passes the tests, it is granted a license and can carry the MOST trademark.

Apart from the legal aspect of granting a license, the Compliance Tests are continuously extended to ensure interoperability between MOST devices and to check their quality. It is not compulsory to use the MOST logo. If an implementer, however, advertises with the MOST logo, it should stand for high quality and interoperability.

The procedure of Compliance Verification is described in chapter 12. It is generally defined by committees within the MOST Cooperation, which are independent of the specification groups. Some persons, however, may have dual functions in the MOST Cooperation, which means they have one function in the specification organization and an additional function in the compliance groups.

2 Survey of the System Architecture

This chapter gives an overview of the system architecture and the basic characteristics of MOST, which are described in more detail in the following chapters.

2.1 MOST Layer Model

When the MOST system was developed, vehicle manufacturers had two basic demands with regard to the functionality of an infotainment bus system:

1. A simple system design based on a function-orientated point of view, and
2. the transmission of streaming and packet data, as well as control information.

The first demand resulted in the development of the MOST framework with *function blocks* being the essential part. Function blocks incorporate all properties and methods of a MOST device (e.g., a CD changer), which are necessary to control the device. Communication with these function blocks takes place over the application protocol, which consists of a self-explanatory mnemonic and does not require an address of the MOST device. Thus, a simple and fast system design of the infotainment domain is possible with a high abstraction level.

The function blocks constitute the interface with the application and therefore belong to layer 7 of the ISO/OSI model (fig. 2.1). They are introduced in section 2.2 together with the application protocol and explained in detail in chapter 3.

The second demand was achieved with the frame structure. It synchronously transports multimedia data, can transmit a great amount of data asynchronously without influencing the synchronous transmission and offers a Control Channel for control commands and status messages. The letter belongs to the application protocol.

The Data Link Layer is based, according to the above mentioned second demand, on a synchronous transmission of data frames. It is realized by the MOST Network Interface Controller. Besides the more common optical bit transmission layer, there is also an electrical physical layer available.

Section 2.4 gives a general survey of the Data Link Layer. This layer is presented in chapter 5. The Physical Layer is introduced in section 2.5. It is described in detail in chapter 6.

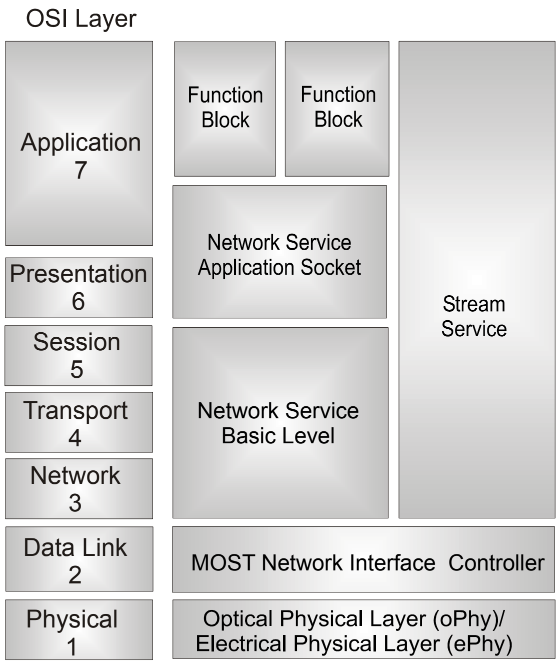

OSI Layer
Application 7
Presentation 6
Session 5
Transport 4
Network 3
Data Link 2
Physical 1
Function Block
Function Block
Network Service Application Socket
Network Service Basic Level
Stream Service
MOST Network Interface Controller
Optical Physical Layer (oPhy)/ Electrical Physical Layer (ePhy)

Fig. 2.1: MOST in the ISO-OSI model

The driver software of the MOST System, i.e., the Network Service, is a middleware between the function blocks and the Data Link Layer. It covers layer 3 up to parts of layer 7 of the ISO/OSI-Model. *NetServices* is the product name used by the company SMSC for the Network Service.

Due to the fact that the MOST system was developed for the synchronous transmission of audio and video data in particular, there are additional Stream Services used for their transport. They are shown in figure 2.1, but do not belong to the actual OSI model.

2.2 Application Framework

The key elements of the Application Framework are the above mentioned function blocks and their dynamic behavior.

A function block is an object for controlling dedicated functions in the MOST network. There are function blocks for controlling applications, such as the CD changer or audio amplifier, and function blocks for the network management. The function blocks provide comprehensive tools for implementing complex audio functions. They make it relatively easy for the application design engineer to create distributed audio applications on a highly abstracted level.

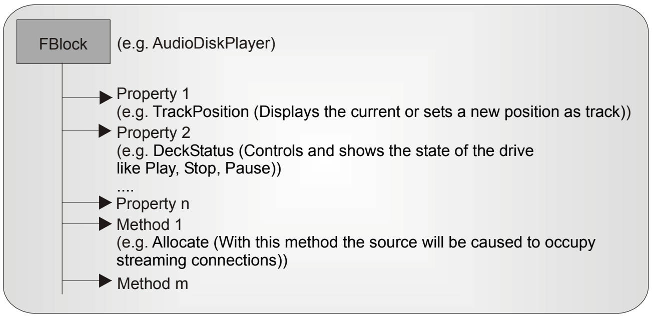

FBlock
(e.g. AudioDiskPlayer)
Property 1
(e.g. TrackPosition (Displays the current or sets a new position as track))
Property 2
(e.g. DeckStatus (Controls and shows the state of the drive like Play, Stop, Pause))
....
Property n
Method 1
(e.g. Allocate (With this method the source will be caused to occupy streaming connections))
Method m

Fig. 2.2: Function block of the CD player (Audio-DiskPlayer)

A function block defines the interface of an application to be controlled. It consists of properties and methods. The properties describe or change the condition of the function to be controlled and the methods execute actions which, after a specified period of time, achieve a result. The MOST specification uses the term "function" to include both properties and methods.

In order to be able to ensure a uniform implementation, every function block has an XML description used as an interchange format. This description can be edited with the MOST Editor (section 11.2.2).

Figure 2.2. shows an example with extracts of the properties and methods of the function block of a CD player (AudioDiskPlayer).

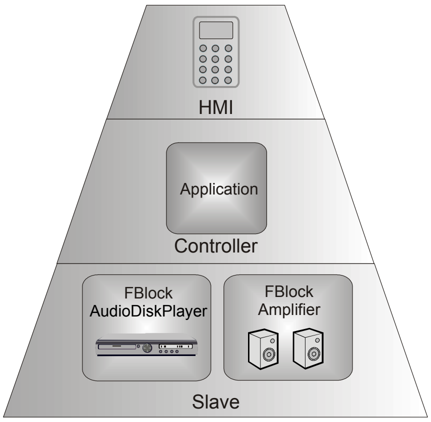


Fig. 2.3: Hierarchy in the MOST system (according to [Tutorial])

The MOST specification differentiates between three views of the interaction with function blocks (fig. 2.3):

- HMI
- Controller
- Slave

A *slave* is a MOST device, which is controlled by a Controller. It provides its functionalities by means of properties and methods of its function blocks. A slave has no system knowledge at all, i.e., information about other devices is not stored statically and consequently the slave does not control other slaves either (fig. 2.4). They can easily be added to or removed from a MOST system, without the software being modified or other slaves being influenced. This means that a CD changer or amplifier, can easily be used in different vehicle platforms, if they are implemented as slaves

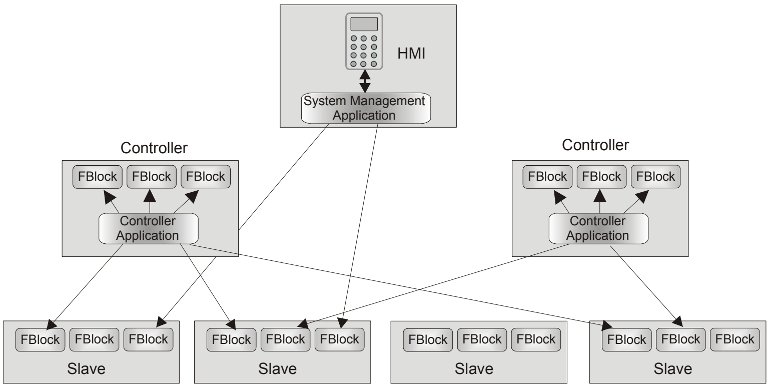


Fig. 2.4: Interactions in the MOST hierarchy (according to [Tutorial])

A *Controller* is an application for administration of a functional part of a MOST system, i.e., it controls the function blocks of one or more slaves (fig. 2.4). A tuner, for example, can control its corresponding amplifier. For this purpose, the Controller requires partial system knowledge, which means that it must know the function blocks to be controlled. A proxy (or stub) for a function block to which the Controller has access is also called the shadow of this function block. Apart from that, a Controller can also have function blocks of its own (see also section 4.1).

The *Human Machine Interface (HMI)* is the interface to the user of the MOST system and thus presents the system function on a high abstraction level. It coordinates the various Controllers.

The Controller uses an application protocol, introduced in section 4.2 (see also fig. 2.5), to control a function block. It is transmitted via the Control Channel or the MOST High Protocol in the Packet Data Channel. The device address of the addressed function block does not need to be known, as it is ascertained by the Network Service (see section 9.3.3).

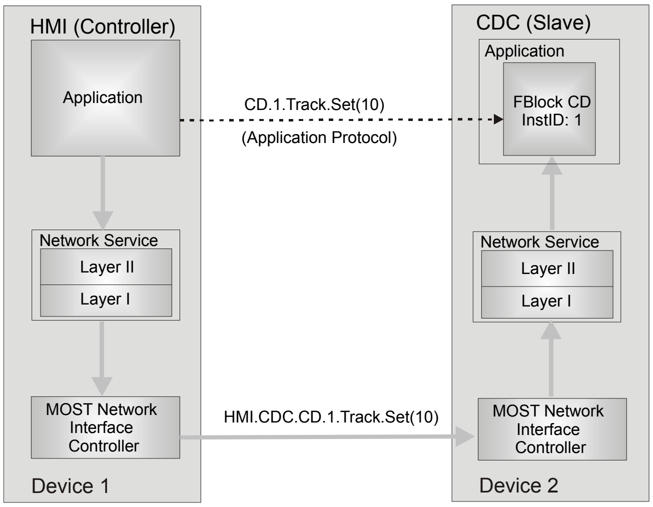


Fig. 2.5: Application protocol (according to [MOST 3.0])

The description language *Message Sequence Chart* (MSC) is used for describing the dynamic behavior and the use of the function blocks. MSC is a technical language which was defined by the International Telecommunications Union (ITU) for specifying and describing the communication behavior of system components among each other and with their surroundings. It is independent of the communication protocol used. The language is introduced in appendix A.

2.3 New isochronous transmission mechanisms for MOST150

MOST150 can handle three different isochronous mechanisms designed for different applications.

The **A/V Packetized Isochronous Streaming** permits transfers of video data streams that arrive without any reference to the MOST time base. The data is already consolidated in small data packets, which can optionally include a time stamp. MPEG data streams that are coded either with variable (VBR) or constant bit rates (CBR) are typical examples of this.

DiscreteFrame Isochronous streams contain data and additional time base information in parallel. The data is arranged in frames. They are not synchronized to the time base of MOST. All PCM streams fall into this category. Here, the PCM data is accompanied by a frame sync signal. An example is the transport of a 44.1 kHz signal over a 48 kHz MOST network.

QoS IP (Streaming) mode is used for packet transmission which allows for full quality of service requirements. Typically, resources on the Packet Data Channel are shared between all applications on the network. Certain scenarios require guaranteed bandwidth. For example, when transporting video data as an IP stream on the MOST Ethernet channel, the occurrence of sudden, high-volume data transfer would lower the available bandwidth for the video signal, affecting the video transmission. If it uses QoS IP (Streaming) mechanism, the sudden data transfer does not interfere with the video transmission, avoiding disturbances.

2.4 Network Services

Network Services are a standardized protocol stack for MOST, mapping level 3 up to level 7 of the OSI model. They are usually divided into two layers. The Network Services also include the transport protocol *MOST High Protocol,* the adaptation layer *MAMAC*, which maps the TCP/IP stack onto MOST (only MOST25 and MOST50) and accordingly a pure TCP/IP stack for MOST150, so-called Ethernet over MOST (see also chapter 5.7).

They are attached to the MOST Network Interface Controller and provide a programming interface for the application, which basically consists of the function blocks (fig. 2.10). They contain modules for the transfer of packet data in the Packet Data Channel and modules for the control of the network via the Control Channel. The Network-Services are implemented on an External Host Controller (EHC). The control of streaming connections is also part of the Network Services. However, the synchronous/isochronous data transmission is not.

2.5 MOST Data Link Layer

The Data Link Layer defines the basic transmission mechanisms of the MOST bus. This includes the definition of the structure of the data frames, as well as the function of the TimingMaster and the TimingSlave.

2.5.1 MOST Frame

The second requirement of the MOST system, i.e., the synchronous transmission of multimedia data, is directly reflected in the frame structure. A frame contains one channel for the synchronous or isochronous (only MOST150) transmission of streaming data, one channel for the asynchronous transmission of packet data, and one channel for the transmission of control data (fig. 2.6). In the Streaming Data Channel, static connections between a streaming source and one or several streaming sinks can be built up with the sampling rates 44.1 kHz (MOST25 and MOST50) and 48 kHz (MOST150).

The control of the connection set-up and disconnection, and the exchange of the control messages for the function blocks are effected via the Control Channel. The data for the commands transmitted via the Control Channel are distributed over several subsequent frames. A MOST25 frame provides two bytes in each frame for that purpose, and both MOST50 and MOST150 frame contain four bytes.

Packet data is transmitted in the Packet Data Channel without influencing the synchronous data transmission at all. It transports configuration data of the navigation system, for example. A data link layer protocol is used for that purpose, which covers the Packet Data Channel. It can transport pure Ethernet Data in MOST150.

The boundary between the Streaming and the Packet Data Channel can be set using the boundary descriptor.

As figure 2.6 shows, a MOST25 frame contains 64 bytes in total. MOST50 can transport 128 byte at the same time due to the double bandwidth, and MOST150 carries 384 bytes.

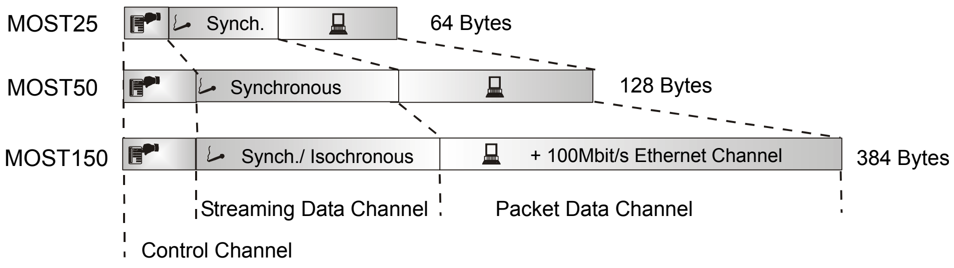


Fig. 2.6: Basic structure of a MOST frame

Chapter 5 gives a detailed introduction of the MOST protocols of the Data Link Layer.

2.5.2 TimingMaster, TimingSlave

As MOST is a point to multi-point data flow system (i.e., the streaming data has a source and any desired number of sinks), all devices share a common system clock pulse derived from the data stream. They are thus in phase with each other and can transmit all data synchronously, which makes any mechanisms for signal buffering and signal processing redundant.

The system clock is generated by the TimingMaster, which is integrated in one of the nodes. In a vehicle, it is usually located in the head unit of the infotainment system. All other nodes are synchronized onto this system clock pulse by means of a PLL connection and are thus referred to as TimingSlaves (fig. 2.7).

When the TimingMaster receives the frame again when it has traveled around the ring, it reclaims the signals by means of a PLL connection and subsequently generates the next frame.

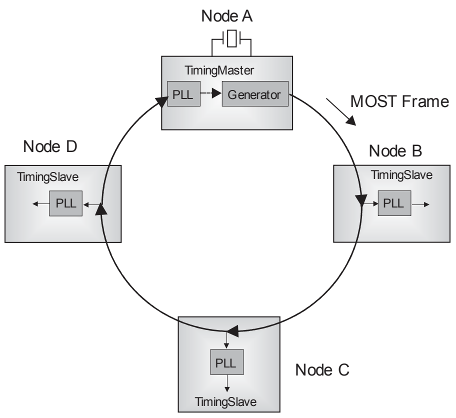


Fig. 2.7: MOST ring with a TimingMaster and TimingSlaves

Lock/Unlock

A TimingSlave is in a lock state if it receives a signal at the input onto which it can synchronize with the PLL; if not, it goes into unlock state.

The TimingMaster is in lock state if it can regenerate the frame from the signal that has traveled around the ring.

In the case of an optical Physical Layer, the lock state implies that the light is on in the ring (Light On).

A more detailed description is given in section 7.5.1.

2.6 Physical Layer

Table 2.1 summarizes the most important parameters of the Physical Layer. MOST25 stands for a bit rate of the data stream of about 25 Mbit/s and uses an optical Physical Layer. The exact data rate depends on the sampling rate of the system. At a sampling rate of 44.1 kHz, the MOST frame is transmitted 44,100 times per second, which results in a data rate of 22.58 Mbit/s at a frame length of 512 bits (see section 6.5.1). This also applies to MOST50. The frame here has a length of 1,024 bits, which results in twice the data rate at the same sampling rate. An electrical Physical Layer was additionally specified for MOST50.

The first generation of the Network Interface Controllers (NIC) can only be used for MOST25. The second generation Intelligent Network Interface Controllers (INICs) are available for MOST25, MOST50 and MOST150. In chapter 10, both generations of MOST Network Interface Controllers are introduced.

	MOST25	*MOST50*	*MOST150*
Bit rate	*circa 25 Mbit/s*	*circa 50 Mbit/s*	*circa 150 Mbit/s*
Physical Layer	Optical	Electrical Optical	Electrical (based on coax line drivers) Optical
NIC	OS8104; OS8104A	-	-
INIC	OS81050; OS81060	OS81082; OS81092	OS81110; OS81120

Table 2.1: Important parameters of the Physical Layer

2.6.1 Optical Physical Layer

MOST uses Plastic Optical Fiber (POF) as physical transmission medium and has therefore very good EMC properties.

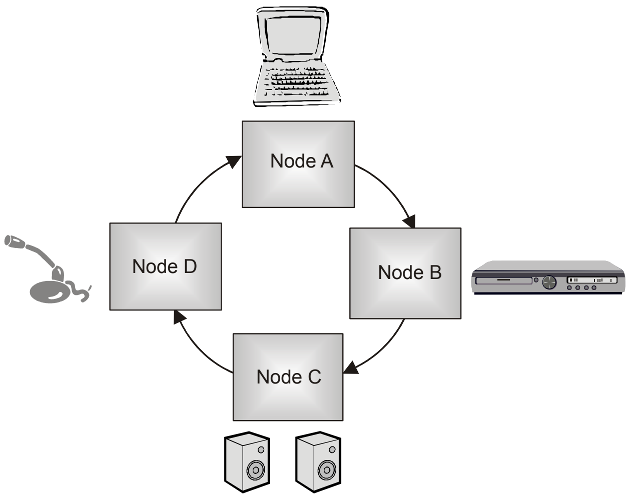


Fig. 2.8: Simple MOST ring

The basic architecture of a MOST network is a logical ring, which transmits the data from one device to another. The logical ring structure is usually realized as a physical ring (fig. 2.8). This is, however, not mandatory. A combined ring and star topology, for example, can be realized with one hub. In a star topology nodes can simply be added or removed without the rest of the network being influenced, which increases the system reliability.

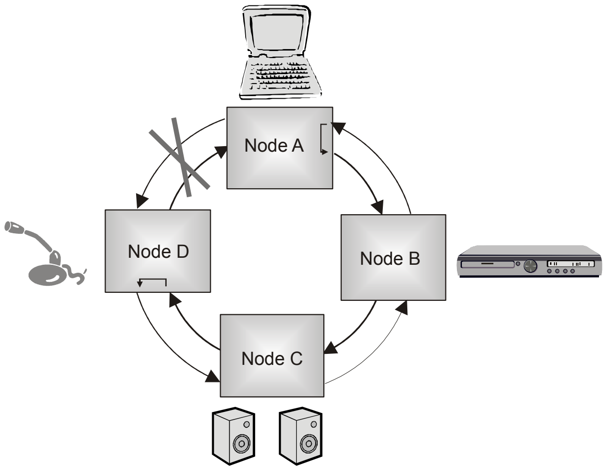


Fig. 2.9: Double ring struture

The principle of transmitting the data stream from one node to the next, which can be arranged in a star topology or any possible mixed form of ring and star, is always maintained.

In order to make MOST highly available in the case of errors, a double ring can be realized. For this purpose each node must have two optical receivers and two transmitters (see fig. 2.9). If there is a problem between two nodes, the ring can be closed via the redundant segments. The Bosch conference system *Praesideo* [Praesideo] uses such a double ring structure. It is employed in big complexes of buildings and complies with the European Safety Standard EN 60849.

The optical Physical Layer is introduced in detail in chapter 6.

2.6.2 Electrical Physical Layer

The electrical Physical Layers additionally specified for MOST50 and MOST150 based on coax line drivers are described in section 6.6.

2.7 MOST Device

The basic architecture of a MOST device, as it is shown in figure 2.10, can be deduced from figure 2.1. It shows a device with an optical Physical Layer (oPhy). The Fiber Optic Receiver (FOR) transforms the light signal into an electrical signal, which is transmitted to the Rx input of the Interface Controller. The Tx signal from the controller is retransformed into a light signal by the Fiber Optic Transmitter (FOX). Receiver and transmitter are called uniformly Fiber Optic Transceivers (FOT) or the physical interface.

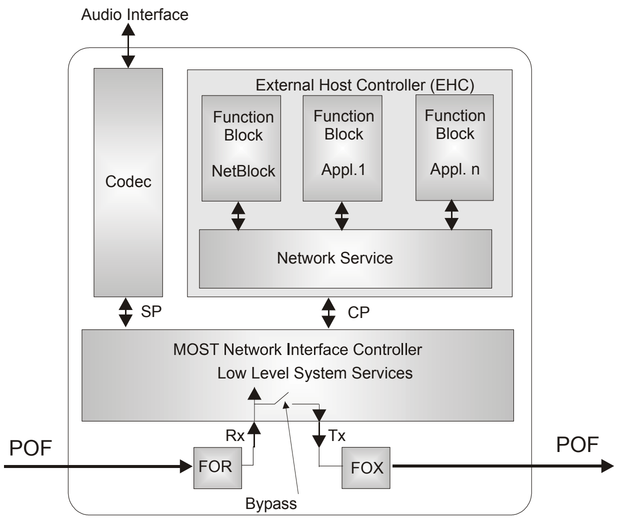


Fig. 2.10: MOST device model with an optical Physical Layer

The Data Link Layer is realized in the MOST Network Interface Controller, which controls the access to the three different channels. The Network Service and the function blocks are implemented on a microcontroller, which is also referred to as *External Host Controller* (EHC). The access from the EHC to the Packet Data Channel and the Control Channel is effected via the Control Port (CP) of the MOST Network Interface Controller.

Chapter 9 introduces the Network Service.

For the streaming services, a processor-free codec can be used in the simplest case or a signal processor in a high-grade system. The interface with the MOST Network Interface Controller is the Source Port (SP) (in the case of the NIC) or the Streaming Port (in the case of the INIC). In addition to this, the INIC offers a universal, efficient interface through MediaLB, which can address all channels (see section 10.11).

The MOST Network Interface Controller has an integrated bypass. It is closed (activated) or remains closed, if the controller is not in normal operation mode, and then transmits the incoming data stream nearly without delay and regeneration via Tx. This applies, for example, in the case of errors or when the device is started up.

2.8 Network Management

In addition to function blocks for application functions (such as CD changer or tuner), there are also a number of function blocks for network management functions, such as the *NetBlock, PowerMaster, NetworkMaster* and the *ConnectionMaster.* They are introduced in chapter 7.

The *NetBlock* must be implemented in every device. The other function blocks for network management are found only once in the system. The specification does not regulate, on which device these functions are implemented. They may be distributed in the system, but are usually found together in the head unit.

2.8.1 NetBlock

The *NetBlock* is responsible for the administration of a device. It has, for example, a list of all the function blocks implemented on the device, and manages all addresses of the device (node position address, logical address, group address). Section 4.3.4 goes into more detail.

2.8.2 PowerMaster

The *PowerMaster* defines the function of a selected device in the MOST network, which is responsible for starting up and shutting down the MOST network and supervises the status of the power supply. It does not have a function block specification of its own, but a function block identifier for identifying the device. The function block *PowerMaster* is introduced in section 4.3.4.

The network can be woken up by any node. A wake-up which was not initiated by the PowerMaster (e.g., by the gateway), is characterized as a slave wake-up.

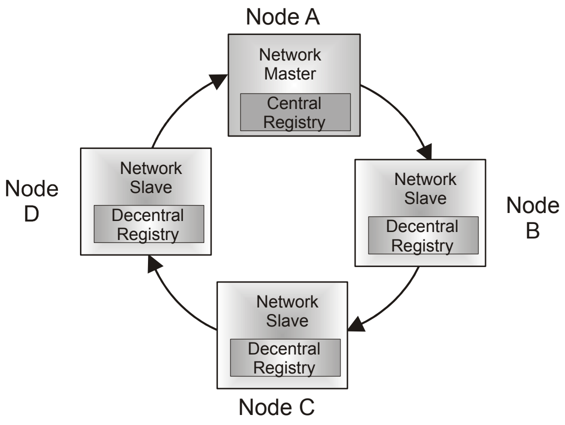


Fig. 2.11: NetworkMaster and NetworkSlave

2.8.3 NetworkMaster; NetworkSlave

The NetworkMaster starts up the system and administers the network status, as well as the Central Registry. The latter lists all function blocks existing in the network, and the respective address of each device, on which a function block is implemented.

All other devices in the ring are called *NetworkSlaves*. They may have a Decentralized Registry, a subset of the Central Registry. It comprises the function blocks of the communication partners of the device.

Figure 2.11 shows a MOST ring with the NetworkMaster and the Slaves.

The NetworkMaster function is the central part of chapter 7.

The respective function block is the *NetworkMaster* (section 4.3.4).

2.8.4 ConnectionMaster

The ConnectionMaster is the function block that contains the interface to the Connection Manager, which is responsible for set-up and disconnection of streaming connections. The implementation of the ConnectionMaster function block is optional (see also chapter 4.3.4).

Figure 2.12 shows the basic operational mode of the ConnectionMaster. An initiator orders the ConnectionMaster to establish a connection. Thereupon it reserves the necessary bandwidth in the synchronous area and reports the channel number of the data source and sink. The disconnection is carried out analogously.

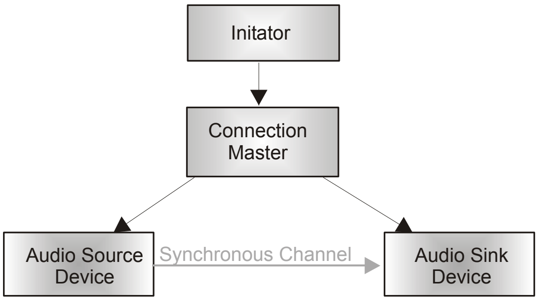


Fig. 2.12: Basic principle of the ConnectionMaster

2.9 Error Management and Diagnosis

MOST specifies a ring break diagnosis. It can locate broken POFs and delivers the result to a dedicated device. It includes also an optional Electrical Line Control. Chapter 8 goes into more detail.

The Diagnostic Protocols Adaption Specification [DPA] defines requirements for the adaption of diagnostic protocols, e.g., UDS (Unified Diagnostic Services) to MOST.

2.10 Transmission of Multimedia Data

For transmitting content through streaming connections, coding procedures have to be defined and the copy protection is to be ensured, particularly for commercial entertainment content, protected by some form of access method.

MOST uses current coding procedures, such as PCM and MPEG, the implementations of which are to be found in the *MOST Specification for Stream Transmission* [Stream].

MOST applies the Digital Transmission Content Protection (DTCP) standard for transmitting content protected by copyright. This technology comes from the *5C Group*, a cooperation of the five device manufacturers *Intel, Matsushita, Toshiba, Sony* and *Hitachi.* It is available on the website of Digital Transmission Licensing Administrator (DTLA) www.dtcp.com. DTCP was developed to ensure the safe transmission of signals from a set-top box to the DVD video recorder. The renowned Hollywood studios Sony and Warner also use this protection scheme in order to transport films in digital networks.

The MOST specification *MOST Content Protection Scheme DTCP Implementation* [ContProt] defines the necessary functions and services to enable the use of this technology.

3 Survey of the MOST Specifications

This chapter gives a survey of the structure and composition of the MOST Specifications. The current status of the specifications in the amended version can be gathered from the updated list of documents of the MOST Cooperation website (http://www.mostcooperation.com).

3.1 Structure

Figure 3.1 reflects the structure of the MOST Framework and illustrates the dependencies of the documents. The MOST Specification is the main document within the MOST Framework. The arrows thus point to the MOST Specification as reference for all other documents, such as the specifications of the MOST function blocks (FBlocks) and the corresponding dynamic specification.

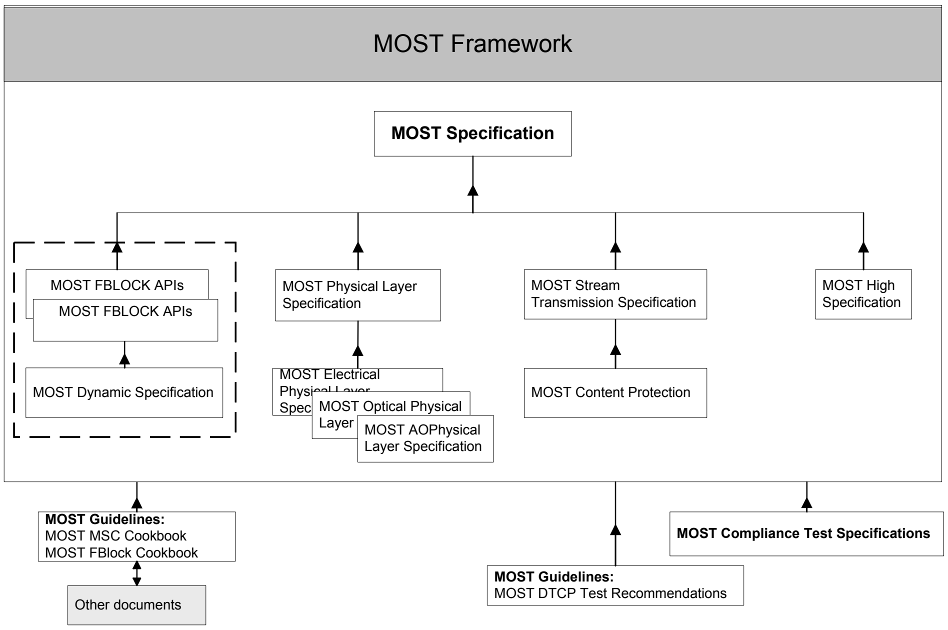

MOST Framework
MOST Specification
MOST FBLOCK APIs
MOST FBLOCK APIs
MOST Dynamic Specification
MOST Physical Layer Specification
MOST Electrical Physical Layer Spec
MOST Optical Physical Layer
MOST AOPhysical Layer Specification
MOST Stream Transmission Specification
MOST Content Protection
MOST High Specification
MOST Guidelines: MOST MSC Cookbook MOST FBlock Cookbook
Other documents
MOST Guidelines: MOST DTCP Test Recommendations
MOST Compliance Test Specifications

Fig. 3.1: Structure of MOST documents

3.2 MOST Specification

The structure of the MOST Specification [MOST 3.0] follows the layer model and basically comprises the following sections:

- Application
- System concept of MOST
- Device model
- Communication between MOST devices
- Function classes
- Network
- Data transport
- Control Channel
- Packet Data Channel
- Synchronous Channel
- Isochronous Channel
- Ethernet Channel
- Dynamic device behavior
- Error management
- Diagnosis

The MOST System uses a function model which allows a high abstraction of the physical network. The communication uses function blocks (FBlocks) and *functions* which can exist in the physical network detached from the physical devices.

Based on the function model, the MOST System has a strict hierarchic structure. The uppermost level is the System Master, with one or more Controllers controlling a number of Slaves. This hierarchic structure concentrates the system knowledge within only a few components: in the System Master and the Controllers.

The System Master, which is usually located in the Human Machine Interface (HMI), controls the overall system on a high abstraction level and provides the user with the system functionality.

A Controller has partial system knowledge. It controls complex functional system extensions and makes them available to the superior level in a simplified form. Every Controller is an “expert” for the handling of specific functionalities, such as all audio functions.

After the introduction of the system concept, the MOST Specification deals with the communication between MOST devices and the function classes.

In the section about the Network, the structure of the MOST telegram and the different data transport concepts (Control Channel, Packet Data, Synchronous, Isochronous and Ethernet Channel) are introduced.

The section about the *Dynamic Device Behavior* explains the system states, the system start-up and the registration of the function blocks in the NetworkMaster. Additional topics are the system wake-up, error management and the diagnosis.

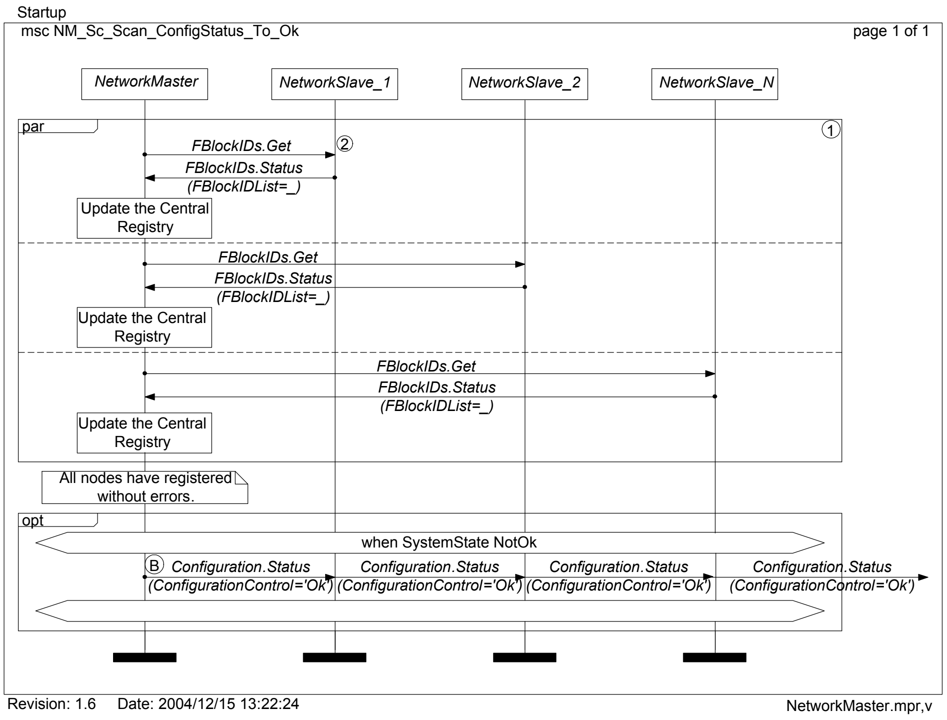

Startup
msc NM_Sc_Scan_ConfigStatus_To_Ok
page 1 of 1
NetworkMaster
NetworkSlave_1
NetworkSlave_2
NetworkSlave_N
par
1
2
FBlockIDs.Get
FBlockIDs.Status (FBlockIDList=_)
Update the Central Registry
FBlockIDs.Get
FBlockIDs.Status (FBlockIDList=_)
Update the Central Registry
FBlockIDs.Get
FBlockIDs.Status (FBlockIDList=_)
Update the Central Registry
All nodes have registered without errors.
opt
when SystemState NotOk
B
Configuration.Status (ConfigurationControl='Ok')
Configuration.Status (ConfigurationControl='Ok')
Configuration.Status (ConfigurationControl='Ok')
Configuration.Status (ConfigurationControl='Ok')
Revision: 1.6 Date: 2004/12/15 13:22:24
NetworkMaster.mpr,v

Fig. 3.2: Exemplary scenario: Regular System Scan

The section *Error Management* describes the behavior in the case of undervoltage, excess temperature and device error and additionally describes system errors, such as the loss of synchronism (Unlock) and the change of registration (Network Change Event) (see section 7.1.2).

The section *Diagnosis* comprises the ring break diagnosis, the Sudden Signal Off behavior as well as the coding error counter. Through these, feature specific automotive requirements are met.

3.3 MOST Dynamic Specification

The MOST Dynamic Specification [DynSpec] is closely connected with the MOST Specification. It takes up the subjects of Network Management, Connection Management and Power Management from the MOST Specification and describes them in the form of Message Sequence Charts (MSCs).

Figure 3.2 shows the example of the scenario *Regular System Scan* with the communication partners NetworkMaster and NetworkSlaves. In the *par* section the NetworkMaster queries, by means of the Control Message `FBlockIDs.Get()`, all MOST devices about the operable function blocks. It thus builds up the Central Registry. The *opt* section describes the case that the NetworkMaster sends the Control Message `Configuration.Status(OK)`, if the network is in the NotOK state (see section 7.1.2).

3.4 MOST Function Catalog

In the MOST function catalog (FBlock Library) all defined function blocks are described. Table 4.7 (section 4.4.1) already gives a survey of the available function blocks.

Every FBlock has its own document, specifying the static and dynamic behavior of the FBlock. The static information structures are described by means of XML. Its structure is defined in a function catalog DTD (Document Type Definition), and is referenced in the DTD Cookbook [DTD Cook]. The dynamic behavior is described in the form of scenarios by means of Message Sequence Charts in accordance with the dynamic specification [DynSpec] (see section 3.2).

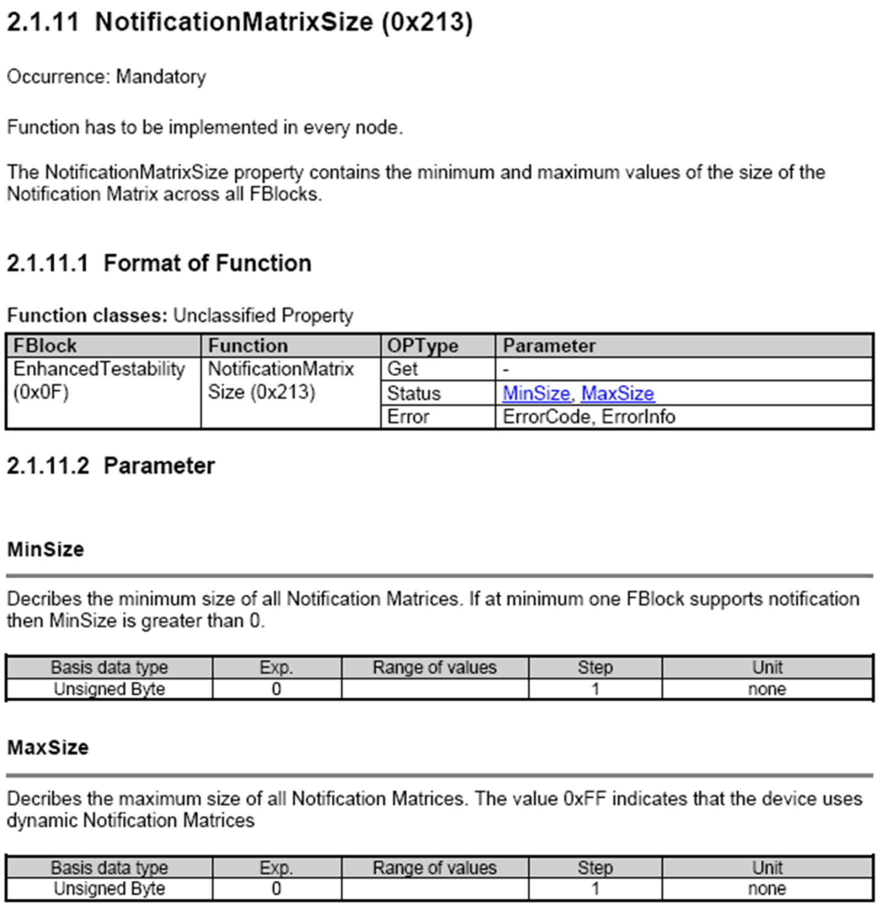
Figure 3.3 gives a survey of the static description mode of the function *NotificationMatrixSize* of the FBlock *EnhancedTestability*. By means of this function, the minimum and maximum values of the size of the Notification Matrix across all FBlocks of a device are given. This function is mandatory, i.e., it must always be implemented. It uses the OpTypes *Get, Status* and *Error*. The parameters *MinSize* and *MaxSize*, used by *Status*, are of the data type *Unsigned Byte.*

3.5 MOST Compliance Test Specifications

According to the structure of the certification test process (chapter 12), the individual scopes are provided with the respective test specifications.

The document *MOST Compliance Requirements* [ComReq] illustrates the general workflow of the certification test process.

The optical Physical Layer Tests are described in the documents *MOST Compliance Test of Physical Layer* [ComPhyL 1.0] and *MOST Compliance Verification Procedure – Physical Layer* [ComPhyLP 1.0] for MOST25 as well as *MOST150 oPhy Measurement Guideline* [ComPhyL 150] and *MOST150 oPhy Compliance Verification Procedure – Physical Layer* [ComPhyLP 150]. Possible corrections or amendments are laid down in the respective *Errata Sheets* ([ComPhyL_Err] and [ComPhyLP_Err]). The tests for the electrical physical layer are specified in the document *MOST ePhy Compliance Test Specification* (with ePhy Compliance Test Patterns) [Com_ePhyL].



2.1.11 NotificationMatrixSize (0x213)

Occurrence: Mandatory

Function has to be implemented in every node.

The NotificationMatrixSize property contains the minimum and maximum values of the size of the Notification Matrix across all FBlocks.

2.1.11.1 Format of Function

Function classes: Unclassified Property

FBlock	Function	OPType	Parameter
EnhancedTestability (0x0F)	NotificationMatrix Size (0x213)	Get	-
		Status	MinSize, MaxSize
		Error	ErrorCode, ErrorInfo

2.1.11.2 Parameter

MinSize

Decribes the minimum size of all Notification Matrices. If at minimum one FBlock supports notification then MinSize is greater than 0.

Basis data type	Exp.	Range of values	Step	Unit
Unsigned Byte	0		1	none

MaxSize

Decribes the maximum size of all Notification Matrices. The value 0xFF indicates that the device uses dynamic Notification Matrices

Basis data type	Exp.	Range of values	Step	Unit
Unsigned Byte	0		1	none

Fig. 3.3: Example of the static description from the FBlock *EnhancedTestability* [FBET]

The test cases of the Core Compliance scope are specified in the document *MOST Core Compliance Test Specification* [Core] and the errata in [Core_Err].

The document *MOST Profile Compliance Test Specification* [ComProf] describes the generic tests of the Profile Compliance scope.

The tests for the individual function blocks and APIs are dealt with in test specifications of their own, for example in the *MOST Profile ConnectionMaster Compliance Test Specification* for the ConnectionMaster.

3.6 Further Specifications, Guidelines and Cookbooks

The following items also belong to to the specifications of the MOST Framework:

- The Physical Layer Specification
- The MOST High Protocol, defining the transmission of Application Messages via the Packet Data Channel [MHP]
- The MAMAC (MOST Asynchronous Medium Access Control) Specification for transmitting network protocols, such as TCP/IP, IPX, NetBEUI and ARP [MAMAC]
- The MOST Content Protection Specification
- The MOST Stream Transmission Specification, defining the streaming formats for transmission of audio-visual data
- Application Recommendations (e.g. [MOST 2003])
- Guidelines and Cookbooks

Apart from the specifications, it is important to document the collected know-how and procedural methods in an adequate manner. On the one hand, this is accopmplished by Application recommendations which provide, e.g., best practice test procedures by the System Integrators. On the other hand, this is accomplished by means of Guidelines and Cookbooks, e.g., the MSC-Cookbook [MSC] (see appendix A) regarding the use of Message Sequence Charts, the DTD-/XML-Cookbook [DTD Cook] and the FIBEX Cookbook [FBX Cook]. They are a collected wealth of experience and focus the wide stream of possible descriptions to the essential aspects.

4 Application Framework

The MOST Specification defines not only the lower layers of a MOST network, which provide the basics for the transmission of data and for the network management, but also the protocols and mechanisms for implementing applications on top of those. In the MOST Specification the interface of an application is called a function block (FBlock) and can be compared to a profile of the Bluetooth standard (see [Bluetooth]).

The MOST Application Framework permits the creation of applications that are independent of a specific implementation on a certain device and of those of the other applications with which they interact in a MOST system. When using the standardized FBlocks of the MOST Specification re-usable and interoperable applications can be developed. An application, for example, that implements the Human Machine Interface (HMI) for a radio can thus be used with any device which offers the standardized application interface of a radio tuner. Devices of different manufacturers can also be used together, if they implement compatible interfaces.

This chapter describes the aspects of the MOST Specification that define the MOST Application Framework, starting with the description of the device model it is based on. The protocol that applications use to exchange data in a MOST system is described next. Finally, the interfaces necessary for the administrative tasks in a MOST system as well as the interfaces for typical applications in vehicle infotainment systems and examples for their usage are given.

4.1 Device Model

Figure 4.1 shows the model of a MOST device as defined in the MOST Specification. On the hardware level, a device has access to the physical transmission layer via the MOST Network Interface Controller (NIC). The Network Service implements a driver layer that controls the access to the interface controller chip and enables applications to carry out basic functions such as sending and receiving messages.

In the MOST Specification, the interface of an application, which offers a specific functionality (i.e., in this context a group of related functions), is referred to as an FBlock. Each device must implement the FBlock NetBlock, which is required for administrative tasks within the MOST system (see section 4.3.3), and the FBlock EnhancedTestability due to compliance reasons. Additionally, each device can, according to its type, implement one or more FBlocks for the functionalities of corresponding applications. For example, it is quite possible that a device offers the functionality of an amplifier, as well as that of a radio tuner, which are then each represented by a corresponding FBlock in the system.

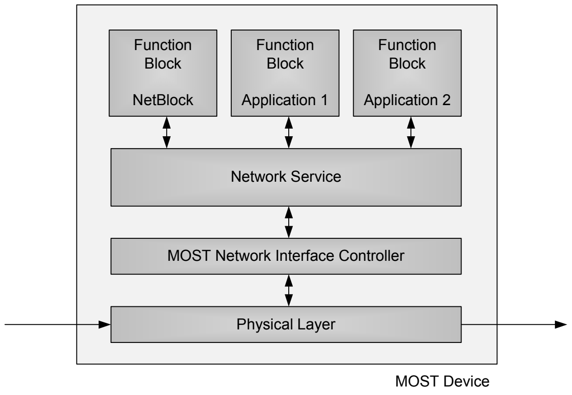

Function Block NetBlock
Function Block Application 1
Function Block Application 2
Network Service
MOST Network Interface Controller
Physical Layer
MOST Device

Fig. 4.1: Model of a MOST device

An FBlock is addressed within a MOST system through its functional address. This ensures that its functionality can be accessed irrespective of the physical location of its functions (i,e., it is not important on which device the functions represented by the FBlock are implemented).

An FBlock, whose functions are used and controlled by an application, is referred to as a Slave. An instance, such as an application, which uses the FBlock, is referred to as its Controller (see fig. 4.2).

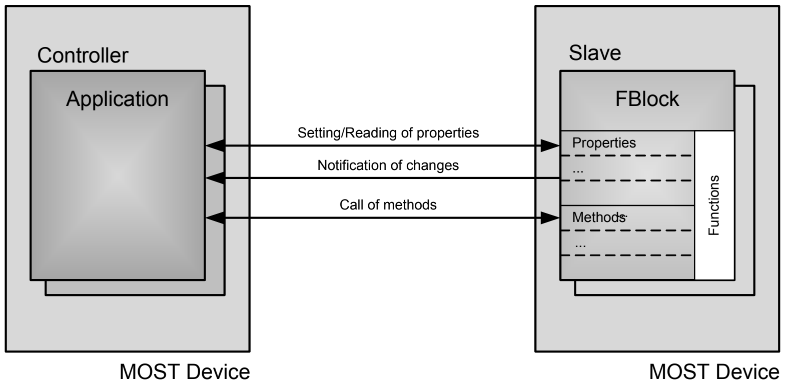


Fig. 4.2: Interacting with an FBlock

Within an FBlock, a differentiation is made between two types of functions. One type is a property that describes a specific attribute of the FBlock (e.g., the currently tuned frequency of a radio tuner), the other a method that triggers a certain action (e.g., a scan for the available radio stations).

A Controller can read the values of the properties of a Slave and, if the respective properties support this, modify them. Additionally, the MOST Specification defines a notification mechanism, by which the Controller registers its interest in particular properties of the Slave and is subsequently informed by the Slave about any changes.

Methods are called by the Controller to trigger a corresponding action of the Slave. After performing the method, the Slave returns the results of the execution to the Controller.

4.2 Application Protocol

As a central aspect of the MOST Application Framework, the MOST Specification defines a communication protocol for the interaction between applications in a MOST system. This application protocol is based and depends on the protocols of the underlying channels. In the early MOST systems, it was typically used with the MOST Control Channel. Because of the increasing size of the transmitted content, currently the Packet Data Channel is also used more frequently.

On the Control Channel, the protocol for sending application messages is called Application Message Service (AMS). On the Packet Data Channel, application messages can be sent using the MOST High Protocol (MHP). These protocols, which handle, for example, segmentation and flow control, are described in detail in chapter 5.

The MOST application protocol determines how to access the services offered by an application in a MOST system. It describes the data format for the messages, which are exchanged for that purpose. It further describes the operational sequences and conditions for read and write access to properties and methods of an FBlock, as well as the use of the notification mechanism.

4.2.1 Data Format

This chapter illustrates the data format of the Application Message Service (AMS) protocol. It consists of the following elements:

[DeviceID.]FBlockID.InstID.FktID.OPType (Data)

When describing communication sequences or examples the above notation is used, in which the individual elements are separated by a dot.

Basically, the data format contains the following three areas:

3. The addressing area consisting of an optional device address (DeviceID), the identifier of the FBlock (FBlockID), and its instance (InstID) in the MOST system.
4. The addressed function with its function identifier (FktID), as well as the operation to be applied to the function (OPType).
5. A data field consisting of a length indication (Length) and the data area for the parameters of the function (Data).

The following table shows an overview of the individual elements and their respective size.

Element	*Size*	*Description*
DeviceID[1]	16 bits	Target device address
FBlockID	8 bits	FBlock identifier
InstID	8 bits	Instance of the FBlock
FktID	12 bits	Function identifier
OPType	4 bits	Operation
Length[2]	16 bits	Length of data field
Data	0 to 65535 bytes	Data field

Table 4.1: Elements of the MOST Application Protocol

Device Address (DeviceID)

The device address (DeviceID) has a size of 16 bits and describes, depending on the context, the receiver, a group of receivers, or the sender of a message.

When addressing individual devices on application level the logical device address is used. Addressing with the node position address is used only for network management or debugging. By using a group address several devices of a MOST system can be addressed simultaneously. The broadcast addresses 0x03C8 (blocking) and 0x03FF (unblocking) are used to address all devices in a MOST system. The device address and the different modes of addressing will be described in detail in chapter 5 *MOST Protocols*.

[1] The DeviceID is not used in the higher layers of the MOST Network Service (Layer II), since FBlocks are only addressed via their functional address.

[2] The length is not transmitted directly, but is based on the underlying protocol.

When programming on application level, the device address is often not used. Addressing can take place here using only the functional address (FBlockID and InstID). If an application does not know the network address of its communication partner (which is usually the case), it can insert the wildcard 0xFFFF for reference into a message to be sent and pass the message on to the Network Service. The Network Service determines the address of the communication partner and replaces the wildcard with it before sending the message (see section 9.2.3).

FBlock Identifier (FBlockID)

The FBlockID describes the type of FBlock and has a size of 8 bits. The FBlockIDs below 0xA0 are defined by the MOST Specification, with specific FBlockIDs reserved for FBlocks required for administrative tasks, which are called system FBlocks, or characterizing specific device functions or device types (see table 4.2). Sections 4.3 and 4.4 of this chapter deal with the system FBlocks and application FBlocks standardized by the MOST Cooperation.

In other areas, the system integrator (FBlockIDs from 0xA0 to 0xEF without 0xC8) or the manufacturer of the device (FBlockIDs from 0xF0 to 0xFE without 0xFC) can define proprietary FBlocks. The FBlockID 0xFF is used as a wildcard, in order to address all FBlocks in a device except for the NetBlock.

FBlock name or area	*FBlockID(s)*	*Description*
Reserved	0x00	Reserved for use by the Network Service
NetBlock	0x01	System FBlock, implemented by each device
NetworkMaster	0x02	System FBlock for the NetworkMaster, implemented by one device in the system
ConnectionMaster	0x03	System FBlock for the ConnectionMaster, optionally implemented by one device in the system
PowerMaster	0x04	Indicates the PowerMaster in the system
EnhancedTestability	0x0F	Implemented by each device
Reserved	0x70 to 0x9F	Reserved for future use
System specific	0xA0 to 0xEF	Assigned by the System Integrator (0xC8 is reserved)
Supplier specific	0xF0 to 0xFB	Assigned by the device manufacturer
Reserved	0xFC	
Supplier specific	0xFD, 0xFE	Assigned by the device manufacturer
Wildcard, All	0xFF	FBlock broadcast

Table 4.2: Structure of the FBlockID area

Instances of FBlocks (InstID)

Since several FBlocks of a certain type may be used in a MOST system, such as an integrated CD drive in addition to an external CD changer, a further differentiation is made between their respective instances via the *InstID*. The InstID has a size of 8 bits; its default value is 0x01. FBlockID and InstID together are called the functional address of an FBlock.

The combination of FBlockID and InstID must be unambiguous throughout a system. For its own FBlocks, a device itself is responsible for the suitable allocation of the InstIDs, for example by putting them into ascending numerical order starting with 0x01. The NetworkMaster is responsible for the functional addresses being unambiguous throughout the system (see section 4.3.4). For example, two different physical devices implement a CD player with the FBlock AudioDiskPlayer and each initializes the InstID with 0x01. This means that the functional address 0x31.0x01 would exist twice in the MOST system. In the case of such a conflict, the NetworkMaster must reset an InstID. As an alternative to the dynamic handling of InstIDs, they can be preset statically by the System Integrator.

The InstID values 0x00 and 0xFF have a special function as wildcards, in case a device implements several FBlocks of a specific type. 0x00 addresses any instance of the corresponding FBlock on a given device (*don´t care*). 0xFF addresses all instances on the device (*broadcast*). Wildcards can only be used in requests. The response messages must include the correct InstID of the answering FBlock.

For the case of system FBlocks, the MOST Specification requires a special handling of InstIDs. For those FBlocks that must be implemented by every device (i.e., NetBlock and EnhancedTestability), the InstID corresponds to the ring position. Starting with 0x00 for the TimingMaster, it is increased by one for each device according to its position in the ring.

The FBlock NetworkMaster, where only one instance is allowed in the system, defaults to InstID 0x01; the InstID 0x00 is also allowed. In the case of requests, the wildcard 0x00 must be used.

Function Identifier (FktID)

Individual functions of an FBlock are distinguished from one another through the *FktID*, which has a size of 12 bits. A function can either be a property (i.e., a variable of the FBlock) or a method. Whether a function is a property or a method, which operations it supports, and which parameters are used (see the following sections), is defined by its interface description in the FBlock specification.

The differentiation between properties and methods is discussed in detail in section 4.2.2, which also deals with the use of the respective function types and their usage.

In the case of the FktIDs, the following subareas are differentiated (see table 4.3). Functions from the “Coordination” area (FktIDs 0x000 to 0x1FF) must be implemented by all MOST devices in the same manner. This area is coordinated by the MOST Cooperation via the GeneralFBlock specification (see section 4.4.2). The “Application” area (FktIDs 0x200 to 0x9FF) defines functions that represent the main application for a specific FBlock according to its FBlock specification. The unique area (0xA00 to 0xBFF) contains functions, which may only exist once in a MOST system. In this case, the System Integrator has to take care to coordinate the usage throughout the whole system.

Similar to the FBlockIDs, the System Integrator (FktIDs 0xC00 to 0xEFF) or the device manufacturer (FktIDs 0xF00 to 0xFFE) can specify proprietary functions.

Name	*FktID area*	*Description*
Coordination	0x000 to 0x1FF	Functions for administrative purposes in an FBlock that are coordinated by the MOST Cooperation
Application	0x200 to 0x9FF	Functions that represent the main application functionality of the FBlock
Unique	0xA00 to 0xBFF	Functions that are defined unambiguously in the entire system
System specific	0xC00 to 0xEFF	Functions that can be used by a System Integrator
Supplier specific	0xF00 to 0xFFE	Functions that can be used by a device manufacturer

Table 4.3: Structure of the FktID area

Operation Type (OPType)

Depending on the type of function (property or method) different basic operations can be applied to a function. Operations are the basic elements of the interaction patterns for properties and methods that or defined by the application protocol. An operation is specified by the *OPType,* which has a size of 4 bits.

The following table shows the total of 16 individual operations. They have different meanings, depending on the function being a property or a method of the FBlock. The first nine operations describe a request to the function and are thus used in messages that are sent by a Controller to a Slave. The last seven operations describe the answers of the Slave and are used in the corresponding reports.

	OPType of Properties	*OPType of Methods*	*OPType*
Requests:	Set	*Start (deprecated)*[1]	0x0
	Get	*Abort (deprecated)*[1]	0x1
	SetGet	*StartResult (deprecated)*[1]	0x2
	Increment	*Reserved*	0x3
	Decrement	*Reserved*	0x4
	GetInterface[2]	GetInterface[2]	0x5
	-	StartResultAck	0x6
	-	AbortAck	0x7
	-	StartAck	0x8
Responses:	ErrorAck[3]	ErrorAck	0x9
	-	ProcessingAck	0xA
	Reserved	*Processing (deprecated)*[1]	0xB
	Status	*Result (deprecated)*[1]	0xC
	-	ResultAck	0xD
	Interface[2]	Interface[2]	0xE
	Error	Error[4]	0xF

Table 4.4: Operation types

The meaning of the different OPTypes and their use in the operational sequences will be explained in section 4.2.2.

Size of the Data Area (Length)

The element *Length* indicates the size of the data area in bytes and has a size of 16 bits. Therefore, a maximum of 65,535 data bytes is possible, with the length 0 designating an empty data field.

The element Length is not transmitted directly, but is calculated by the Network Service using the information of the underlying protocol layer depending on the protocol (e.g., AMS or MOST High Protocol). For example, when the AMS is used,

[1] Since MOST Specification 3.0 all OPTypes for methods without a SenderHandle parameter are deprecated.

[2] The OPTypes Interface and GetInterface are optional.

[3] The OPType ErrorAck may be returned by a property if by mistake an Ack-OPType has been used in a request.

[4] The OPType Error may be returned by a method, if the SenderHandle is not applicable or unknown.

the length is calculated from the number and the length of the individual telegrams on the Control Channel.

Data and Parameter Types

If a function has one or more parameters, the corresponding parameter values are transmitted in the data field *Data*. The types and sequence of the parameters are predetermined by the specification of the function's interface in the FBlock specification.

The MOST Specification defines the parameter types indicated in the following table:

Parameter	*Size*	*Description*
Boolean	1 byte	Boolean value (true or false)
BitField	1, 2, 4, or 8 bytes	Bit field with Mask
Enum	1 byte	Enumeration
Unsigned Byte	1 byte	Integer value (without sign)
Signed Byte	1 byte	Signed integer value
Unsigned Word	2 bytes	Integer value (without sign)
Signed Word	2 bytes	Signed integer value
Unsigned Long	4 bytes	Integer value (without sign)
Signed Long	4 bytes	Signed integer value
String	Variable	Null terminated string with encoding information
Stream	Variable	A stream of arbitrary data
Classified Stream	Variable	A stream with type information
Short Stream	Variable (up to 255 bytes)	Short stream with length information

Table 4.5: Parameter types of the MOST application protocol

The individual parameter types are summarized below. For a more detailed description please see the MOST Specification [MOST 3.0]. The characteristics of the individual parameter types are similar to parameters used in other programming languages.

- **Boolean**: A parameter of type Boolean can have one of the values True or False. Only the lowest-value bit of the respective byte is used.
- **BitField**: The parameter type BitField specifies a bit sequence that can have a total size of 1, 2, 4 or 8 bytes, as required. According to the MOST Specification the first half of a parameter of type BitField is a mask and the second half its data. When performing an operation, the bits of the mask indicate on which

of the corresponding bits of the data the operation is to be applied. In practice, this parameter type is sometimes also used without a mask.

- **Enum**: A parameter of type Enum can adopt one of a maximum of 256 possible pre-defined values. The individual values and their meanings are determined by the function's interface definition.
- **Signed/Unsigned Byte, Signed/Unsigned Word, Signed/Unsigned Long** (integer values): The MOST Specification supports the indicated integer values with a size of 1, 2, or 4 bytes, each with and without sign. For numerical values with signs, the sign is determined by the most significant bit and the value is coded in two's-complement. (The MOST Specification up to version 3.0 does not provide special parameter types for floating-point numbers. Instead, the decimal place for the number parameters can be defined by indicating an exponent value in the interface definition.)
- **String**: For the transmission of character strings, the parameter type String is used. The first byte of the string indicates its coding, e.g. 0x00 for UTF16 and 0x01 for ASCII. It is followed by a null-terminated character string, in which 1, 2 or more bytes are used for each character depending on the coding. The end of the character string is characterized by a null byte; two null bytes in the case of codings with two characters (such as UTF16).
- **Stream**: The parameter Stream defines an arbitrary byte sequence and is used if no suitable description by the other parameter types is possible or if a dynamic data structure is required. If the parameter type Stream is intended to be used together with other parameters, only one of the parameters may be a Stream and it has to be the last parameter.
- **Short Stream**: The Short Stream is intended for representing a short byte sequence. It has a maximum length of 256 characters, with the first character indicating the length of the subsequent byte sequence (255 bytes at most). Because of the length information it does not have the same restrictions as the parameter Stream and can be used anywhere in the sequence of a function's parameters.
- **Classified Stream**: In the case of the Classified Stream, the first two bytes indicate the length of the stream and a subsequent null-terminated ASCII string (without coding information) indicates the media type. The media type corresponds to the ones in the HTTP1.1 specification (see [RFC 2616]).

To handle more flexible data structures, the concept of **stream cases** and **stream signals** has been introduced. Both are defined in the function's interface definition.

With stream cases and stream signals, a parameter of type stream can be structured in two different ways:

- With stream cases a parameter of type stream is further structured into different sequences of stream parameters depending on the value of a selector, similar to the concept of a variant record in programming languages. The selector is a parameter of any type that has to occur before the stream. Often, parameters of type Enum are used as selectors. Depending on certain predefined values of the selector, the stream will consist of a sequence of stream parameters with different types and numbers.
- With stream signals, a stream is divided into different consecutive named elements, each with an arbitrary number of bits.

For details on stream cases and stream signals see the MOST Specification [MOST 3.0].

4.2.2 Dynamic Behavior

After describing the message format of the MOST application protocol and its individual elements, this section deals with its use and the resulting message sequences.

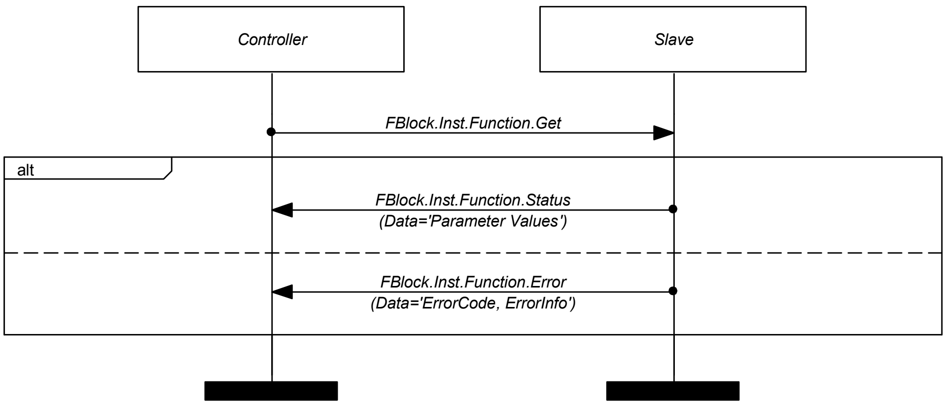

Controller
Slave
FBlock.Inst.Function.Get
alt
FBlock.Inst.Function.Status
(Data='Parameter Values')
FBlock.Inst.Function.Error
(Data='ErrorCode, ErrorInfo')

Fig.4.3: Querying the values of a property with the operation Get

Querying and Setting of Properties

As already explained, a function is either a property or a method. A property describes a specific variable of an FBlock and consists of one or more parameter values. The values of all properties of an FBlock indicate its current status. The values of a property can be queried or set via the MOST application protocol. In addition to that, the MOST Specification specifies a notification mechanism by means of which a device, which is interested in the value of a specific property, is automatically informed of its changes.

To query the current status of a property, a control message with the operation **Get** (OPType: 0x1) is sent to the corresponding function of the FBlock. If no error occurs, the latter answers with a message containing the operation type **Status** (OPType: 0xC) and the current value of the property in the data field. Otherwise, an error message of the operation type Error (0xF) is sent back (see the following section on error handling).

Figure 4.3 shows the corresponding sequence as a Message Sequence Chart (MSC) in MSC 2000 notation [Z.120 99]. This notation is used by the MOST Cooperation and we will also use it for describing the following examples. For a description of the use of MSCs within the MOST Specification, see appendix A and [MSC].

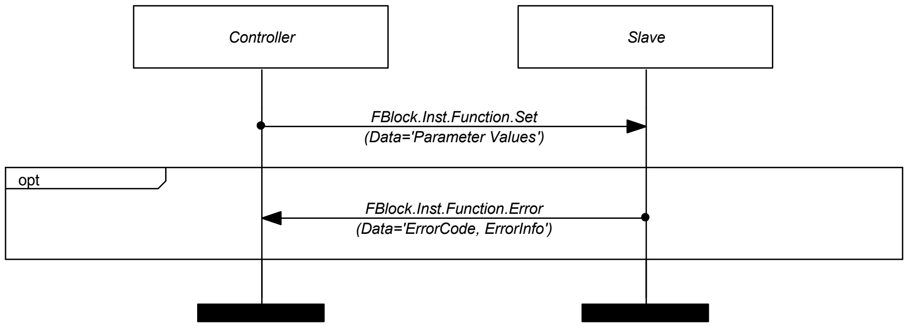

Controller
Slave
FBlock.Inst.Function.Set
(Data='Parameter Values')
opt
FBlock.Inst.Function.Error
(Data='ErrorCode, ErrorInfo')

Fig.4.4: Setting the value of a property with the operation Set

The operation **Set** (OPType: 0x0) is used when no feedback is necessary after the value of a property is set (see fig. 4.4). Only if an error occurs during the process does the FBlock answer with the corresponding error information. The operation Set should be used when the Controller has set a notification for the corresponding property (see the following section concerning the notification mechanism).

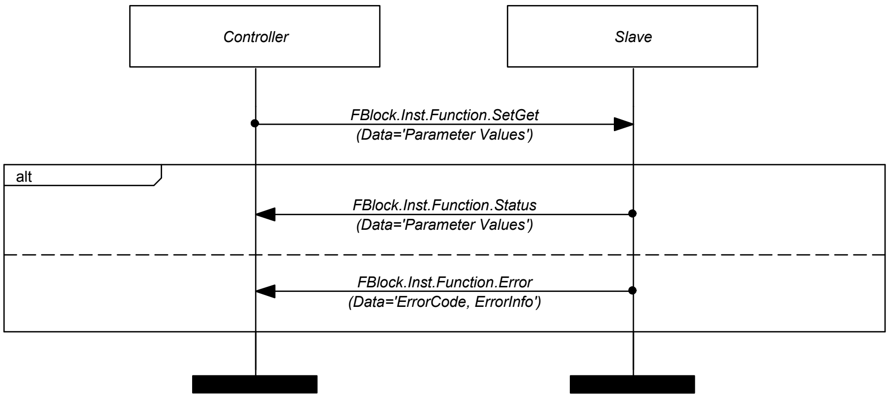

Controller
Slave
FBlock.Inst.Function.SetGet
(Data='Parameter Values')
alt
FBlock.Inst.Function.Status
(Data='Parameter Values')
FBlock.Inst.Function.Error
(Data='ErrorCode, ErrorInfo')

Fig.4.5: Setting the value of a property with the operation SetGet

The **SetGet** operation (OPType: 0x2) is similar to Set, with the difference that a status message with the resulting new value of the property is sent back as confirmation that the change was carried out (see fig. 4.5). This way of accessing properties is to be preferred when a notification for the property is not used at the same time.

Additionally, the MOST Specification defines the operations **Increment** (OPType: 0x3) and **Decrement** (OPType: 0x4), by means of which the value of a property can be increased or decreased by an amount specified in its interface specification.

These operations, however, are less commonly used compared to the other ones previously described. Also, they are more prone to errors, as an unintentional retry increases or decreases the value by two steps instead of one.

Executing Methods

Methods can be used to start a process that changes the status of an FBlock, or represents a long-lasting process (e.g., starting the station scan in the case of a radio receiver).

A Controller can start and, if necessary, abort methods. When starting the method, a sequence of parameters is optionally added which influences the behavior of the method (such as the direction of the station scan). The results which are possibly generated by the method (in this case, for example, the list of the stations) are then returned to the initiator.

In order to differentiate between Controllers within a device (e.g., different applications) that have started a given method, all operations for methods additionally include a special parameter called SenderHandle with a size of 16 bits. This parameter is always returned by the called method together with the answer and enables the sender to associate the answer message with its command message. Since MOST

Specification 3.0, the corresponding operation types without SenderHandle and without the suffix “Ack” are not used anymore.

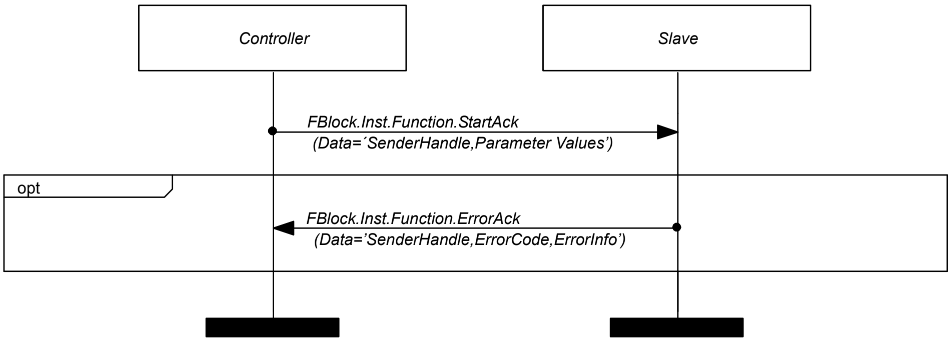

Controller
Slave
FBlock.Inst.Function.StartAck
(Data=´SenderHandle,Parameter Values')
opt
FBlock.Inst.Function.ErrorAck
(Data='SenderHandle,ErrorCode,ErrorInfo')

Fig. 4.6: Starting a method with the operation StartAck

Operation **StartAck** (OPType: 0x8) is used to start a method for which a direct result is not expected or which triggers a continuous procedure (see fig. 4.6). This corresponds to the operation Set for properties and also has the same OPType. For StartAck, the only a feedback is in the form of an error message if an error has occurred.

Methods which generate a result and report it back to the initiator, are triggered by means of the operation **StartResultAck** (OPType: 0x6) (see fig. 4.7). The initiator is informed of the results of the execution by means of a message with the operation **ResultAck** (OPType: 0xD), if no error has occurred during the execution.

If the calculation of the result takes longer (more than 100 ms), the initiator is informed every 100 ms[1] of the ongoing calculation by means of a **ProcessingAck** message (OPType: 0xA). Depending on the method, it is also possible that intermediate results are transmitted with the processing message to indicate the progress of the function execution. The correct handling of processing messages and the appropriate error handling is important to guarantee the responsiveness of an application when implementing interactive HMIs.

[1] The timings described here are standard values as defined by the MOST Specification. However, they can be defined differently for individual methods or by the system integrator for the whole system.

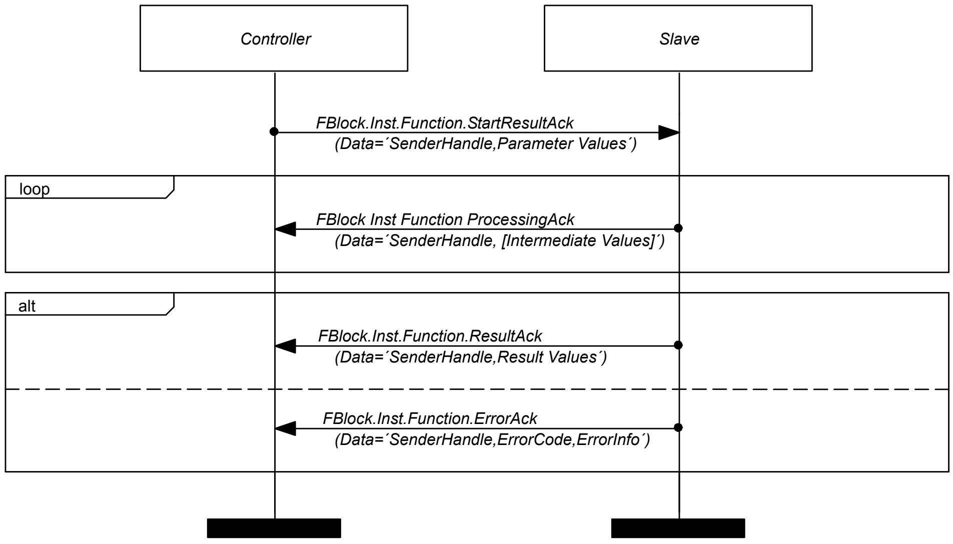

Controller
Slave
FBlock.Inst.Function.StartResultAck
(Data=´SenderHandle,Parameter Values´)
loop
FBlock Inst Function ProcessingAck
(Data=´SenderHandle, [Intermediate Values]´)
alt
FBlock.Inst.Function.ResultAck
(Data=´SenderHandle,Result Values´)
FBlock.Inst.Function.ErrorAck
(Data=´SenderHandle,ErrorCode,ErrorInfo´)

Fig. 4.7: Starting a method with the operation StartResultAck

By calling the operation **AbortAck** (OPType: 0x7) on a running method, its current processing can be interrupted. This is signaled by an error message with the error code **Method Aborted**, which is sent both to the initiator of the method and to the sender of the abort request (in case the two are not the same).

Notification Mechanism

The MOST Specification defines a notification mechanism (see fig. 4.8) so that a Controller interested in a specific property does not have to constantly poll its current value using a Get message. When defining an FBlock, it is implicitly determined which properties support a notification mechanism.

For all of its properties supporting notifications, an FBlock administers a table called **Notification Matrix** in which the addresses of those devices are entered that have set a notification for the respective property. As soon as the value of one of these properties changes, the FBlock sends a status message (notification) with the new value of the property to all devices listed in this table.

A Controller sets or deletes a notification by calling the function **Notification** (FktID: 0x401) of the corresponding FBlock. The first parameter (Control) indicates the operation (see the following listing). In another parameter (DeviceID), the address of the device requesting the notification is indicated. Finally, if required, the following list of FktIDs describes for which functions a notification is requested.

- **SetAll**: Requests a notification of the indicated device for all the FBlock's properties for which a notification is supported.
- **SetFunction**: Requests a notification of the device for all the FBlock's properties indicated in the subsequent list.

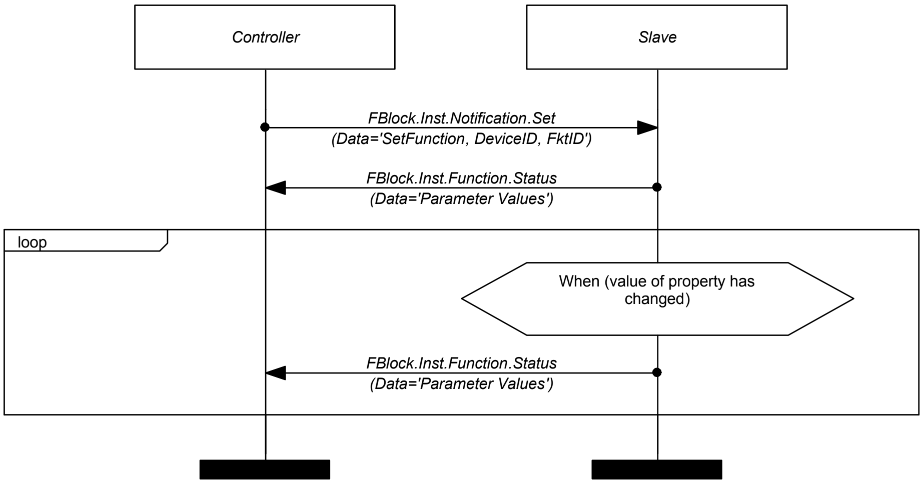

Controller
Slave
FBlock.Inst.Notification.Set
(Data='SetFunction, DeviceID, FktID')
FBlock.Inst.Function.Status
(Data='Parameter Values')
loop
When (value of property has changed)
FBlock.Inst.Function.Status
(Data='Parameter Values')

Fig. 4.8: Notification mechanism

- **ClearAll**: Deletes all notifications of the indicated device for all notified properties of the FBlock.
- **ClearFunction**: Deletes the notification of the device for the properties of the FBlock indicated in the subsequent list.

If a notification of a property is set successfully, the FBlock sends a status message with the current value as confirmation. An example for the use of the notification mechanism is given in section 4.4.4.

Error Name	*ErrorCode*	*Description*
FBlockID not available	0x01	Syntax error: FBlock does not exist
InstID not available	0x02	Syntax error: instance does not exits
FktID not available	0x03	Syntax error: function does not exist
OPType not available	0x04	Syntax error: operation is not defined
Invalid length	0x05	Length of the data field is not identical with the interface specification
Parameter wrong/ out of range	0x06	One or more of the parameters were wrong, i.e., not within the boundaries specified for the function

Error Name	*ErrorCode*	*Description*
Parameter not available	0x07	One or more of the parameters were within the boundaries specified for the function, but are not available at that time
Reserved	0x08	Usage is deprecated
Reserved	0x09	Usage is deprecated
Reserved	0x0A	Usage is deprecated
Device Malfunction	0x0B	General device error
Segmentation error	0x0C	Error during de-segmentation of messages by the AMS
Function specific	0x20	Function specific error
Busy	0x40	Function is available, but is currently being executed
Not available	0x41	Function is implemented in principle, but is not available at the moment
Processing Error	0x42	General error during execution of a method
Method Aborted	0x43	This error code is used to indicate that a method has been aborted by the AbortAck operation
System specific	0xC0 to 0xEF	Error codes that are defined by a System Integrator
Supplier specific	0xF0 to 0xFE	Error codes that are defined by a device manufacturer

Table 4.6: Error codes of the MOST protocol

Error Information

If an error occurs during the execution of a request, whether a property or a method, a message of the operation type **Error** or **ErrorAck** (OPType: 0xF or OPType: 0x9) is returned. The error code (parameter ErrorCode) is transmitted in the first byte of the data field. The second byte contains the parameter ErrorInfo with additional information depending on the error code, which will not be discussed here. For a more detailed description see the MOST Specification [MOST 3.0].

There are some specific requirements concerning error responses in a MOST system. In order to avoid a cyclical sending of error information, error messages may only be sent from the Slave to the Controller (with the exception of segmentation errors).

The general error information shown in table 4.6 is specified by the MOST Specification.

4.3 MOST Function Blocks

This section goes into the details of the structure of an FBlock and describes the important aspects that have to be observed for its definition. Besides the general structure of an FBlock, the concept of Function Classes is described, which provides guidelines for defining the individual functions of an FBlock.

Additionally, the system FBlocks, such as the NetBlock, required for administrative tasks in a MOST system, are introduced in this section. Finally, the functions are described that are required for the administration of streaming connections and for the transmission of multimedia data.

4.3.1 Structure of a Function Block

The FBlock specification defines which functions belong to an FBlock of a specific type. The type of an FBlock is determined by its FBlockID.

The MOST Cooperation standardizes a set of FBlock types, some of which are used for system tasks, others describe certain application functionality. The latter are often extended by device manufacturers and System Integrators to integrate their own proprietary functions. Other FBlock specifications are defined completely by the System Integrator or device manufacturer – either because they define a proprietary functionality or a new functionality for which a standardized FBlock specification does not yet exist. The FBlocks standardized by the MOST Cooperation are collected in an function library (see section 4.4.1).

For all FBlocks, the following functions are required by the MOST Specification:

- **FktIDs** (FktID: 0x000): A list of all functions implemented by the FBlock can be queried by the property FktIDs. In order to compress this possibly long list of FktIDs, a coding is used in which only those FktIDs are included in the list, in which there is a transition between a continuous area of implemented and of non-implemented functions (for details see [MOST 3.0]).
- **FBlockInfo** (FktID: 0x011): This mandatory function returns information about the FBlock's name and version, the version of the MOST Specification the FBlock has been implemented for, the System Integrator in whose system the FBlock will be used, and a string (FBlockType) that is assigned by the System Integrator to identify different instances of the same FBlock in the system (e.g., a left and right display unit for a rear-seat entertainment system).
- **Notification** (FktID: 0x401): The property Notification must be implemented by every FBlock that allows the setting of notifications for one or more of its properties. By means of this property, a device can set or delete notifications (see section 4.2.2).

- **NotificationCheck** (FktID: 0x402): The optional property NotificationCheck can be used to check which notifications are currently set.

In addition to these functions, each FBlock must implement the functions from the coordinated range that are indicated as conditional according to the respective pre-condition defined in the GeneralFBlock specification [GenFB]. Furthermore, all functions defined as mandatory in the corresponding FBlock specification have to be implemented. Functions that are indicated as optional may be omitted.

When defining and deploying a MOST system, the System Integrator indicates in detail the devices that make up the system and the FBlocks which are to be implemented by each device. Usually, the System Integrator also indicates in detail, which functions are to be implemented by the individual FBlocks.

4.3.2 Function Classes

Function classes simplify the specification of an FBlock, by giving specific guidelines concerning the data types and operations of a function. A function class is assigned to a function in the FBlock specification and specifies particular settings for the parameters of a function such as the maximum length of a sequence of characters. Additionally, a function class defines which operations can be executed on the corresponding function. Finally, function classes are the basis for more complex, compound data types, such as records or arrays.

Function classes enable a structured and uniform modeling of functions. Additionally, they support the use of tools for modeling and code generation and simplify test automation.

If a function cannot be assigned to any of the function classes described below, it is assigned to the function class **Unclassified Property** for properties and **Unclassified Method** for methods.

The MOST Specification specifies the following simple function classes for properties, the status of which can be described with a single parameter:

- **Switch**: The function class Switch encapsulates a Boolean parameter and describes a “switch” with two states, which can be set and read.
- **Number**: The function class Number is used for the different parameter types of signed and unsigned numerical values. This function class has the following settings: the parameter type, its maximum and minimum value, an exponent defining the decimal place and the increment, by which the numerical value is changed by Increment and Decrement operations.
- **Text**: The function class Text is based on the parameter type String and defines a sequence of characters. Its settings indicate the maximum length of the string.

- **Enumeration**: Enumeration types are described by the function class Enumeration and are based on an Unsigned Byte. This function class defines how many elements the enumeration type has and what values they can have including an textual description. For example, the function DeckStatus of the FBlock AuxIn can have the values 0x00 for “Play”, 0x01 for “Stop”, 0x02 for “Pause”, etc.
- **BoolField**: The function class BoolField defines a sequence of bits with a length of 8, 16, or 32 bits in total and is based on an Unsigned Byte, Word or Long, respectively. It is also permitted to structure the bit sequence into addressable elements of several consecutive bits. Leading bits, which are not used, remain empty.
- **BitSet**: A property, whose status can be described by a parameter of type BitField, is assigned to the function class BitSet with the given length. As defined for the parameter type BitField, its first half contains the mask and its second half the data.
- **Container**: Finally, the function class Container is used for further unstructured data. This function class is based on a parameter of the type Stream, Classified Stream or Short Stream. Its settings define the maximum allowed length.

Additional function classes are used to represent more complex data structures consisting of several parameters. For these function classes a leading parameter **Position** is used to indicate the individual elements which are to be accessed. Position consists of the two parts **PosX** and **PosY** of one byte each. PosX indicates the position in the outer data structure, PosY the position in a possible inner data structure. In the case of a two-dimensional array, PosX would indicate a certain row and PosY a column. If PosX is not equal to zero and PosY is zero, the entire row of the array is used. With PosX and PosY zero the array is addressed as a whole.

For properties there are the following structured function classes:

- **Record**: A Record consists of an arbitrary sequence of individual elements, each with a certain function class, where individual elements can be accessed by the parameter PosX. Structured function classes, such as an Array, are also allowed as elements (in this case PosY is used to address elements of the array).
- **Array:** An Array describes a sequence of a maximum of 256 elements of the same type, which can also be accessed individually by the parameter PosX. The settings of the function class Array indicate the function class of the individual elements and the maximum length of the Array.
- **DynamicArray**: The DynamicArray is a variant of an Array, in which the individual elements are accessed via a 16-bit tag instead of PosX. In this case, longer arrays are possible. Additionally, functions for inserting and deleting elements are defined. Since the tag of an element, unlike its position, does not change when it is inserted or deleted, the DynamicArray is also well suited for more dynamic data structures.

- **LongArray**: In the case of the LongArray, the individual elements are also accessed via a 16-bit tag. Additionally, a window mechanism is defined, by means of which a Controller defines a specific section (window) of the array. Afterwards it is informed only about changes in this section, which enables an efficient HMI implementation for displaying a list where only the currenty visible section according to the screen size is updated (e.g., for displaying the phone book of a cell phone).
- **Map**: The function class Map is similar to DynamicArray. However, in addition it is able to transmit the insertion or deletion of single or multiple lines via notification. Therefore, updates for a function of class Map can be transmitted very efficiently.
- **Sequence Property**: The function class Sequence Property is a simple function class, by means of which a sequence of individual elements can be set and read. Multi-dimensional data types and the separate access to individual elements are not supported.

For methods only the following two function classes are defined besides Unclassified Method:

- **Trigger**: The function class Trigger describes a function class without parameters, which triggers a specific action.
- **Sequence Method**: The function class Sequence Method is a function class for methods with a list of simple parameters.

4.3.3 Function Interfaces

Since MOST Specification 3.0 [MOST 3.0] the mechanism of function interfaces is optional. Usually, the concept of function classes is only necessary for the definition of a function. Retrieving information about the function interface, i.e. the function class and its settings, during runtime is not common.

The MOST Specification specifies the optional operations **GetInterface** (OPType: 0x5) to request a function's function interface. The result is transmitted via the operation **Interface** (OPType: 0xE). The coding of the relevant information is detailed in the MOST Specification.

4.3.4 System FBlocks

This section gives an overview of the FBlocks which are required by the MOST Specification for the purpose of system administration. As mentioned earlier, there are FBlocks implemented by all devices of a MOST system and FBlocks only provided by one specific device.

NetBlock (FBlockID:0x01)

The NetBlock is responsible for central tasks of the system administration concerning a certain MOST device. It must be implemented by every device in a MOST system.

The most important functions of the NetBlock are described below. The complete interface of NetBlock is specified in [NetBlock]:

- **FBlockIDs** (FktID: 0x000): The function FBlockIDs returns a list of all active application FBlocks that are offered by the device as pairs of FBlockID and InstID (system FBlocks like the NetBlock itself or the FBlock EnhancedTestability are not included). An FBlock may only be included in this function if it is correctly initialized and currently able to offer its functionality to a Controller in the MOST system. Furthermore, the function FBlockIDs can be used to set the InstID of an FBlock. Its main use is during the system scan of the NetworkMaster (see below).
- **DeviceInfo** (FktID: 0x001): Queries characteristic parameters of the device, such as the name of the manufacturer or a serial number.
- Addressing Functions: Further functions provide information about the node position address (**NodePositionAddress**, FktID: 0x002), the logical address (**NodeAddress**, FktID: 0x003), or the group address (**GroupAddress**, FktID: 0x004) of the device.
- **ShutDown** (FktID: 0x006): The shutdown procedure of a MOST system is controlled through this function (see below).
- **ShutDownReason** (FktID: 0x009): After start-up of a MOST system a unique Controller may query the reason for the shutdown from each device. In case of a shutdown because of a failure (sudden signal off or critical unlock), each device that has detected such a failure returns respective information through this function.
- **ImplFBlockIDs** (FktID: 0x012): This optional function returns the list of FBlockIDs of all FBlocks implemented by the device. In contrast to the function FBlockIDs, also the FBlocks that are not currently available can be queried.
- **RBDResult** (FktID: 0x405): The result of a ring break diagnosis is transmitted with this function. For details see chapter 8.
- **Boundary** (FktID: 0xA03): This function is only available on a device which acts as the TimingMaster. It enables the administration of the bandwidth allocation on the data link layer (read out and set). The corresponding mechanism is described in section 5.1.

FBlock EnhancedTestability (FBlockID: 0x0F)

The FBlock EnhancedTestability (ET) is necessary to perform the tests required for the Core Compliance of a MOST device (see [Core 3.0]) and must be implemented by each device. These tests check basic system functions of the device, such as the allocation of an address. Details of the functions of FBlock ET will not be given in this context; it is described in detail in [FBET].

The FBlock ET offers functions for test automation. The function *Reset* (FktID: 0x221), for example, can be used to reset the device in order to start from a stable state. It also contains functions for executing a functionality which can otherwise not be triggered automatically, such as the function *AutoWakeup* (FktID: 0x201), which simulates the spontaneous wake-up of the device.

FBlock NetworkMaster (FBlockID: 0x02)

The FBlock NetworkMaster is responsible for the central administration of the system configuration of a MOST system. It is thus mandatory for each MOST system and must be implemented by a designated device. Often, a specific device is responsible for all master functionalities in a MOST system, which means that the device is usually not only NetworkMaster, but also TimingMaster, ConnectionMaster and PowerMaster. However, except for NetworkMaster and TimingMaster, this is not mandatory.

The NetworkMaster administers the system configuration and thus the current system status of the MOST system. The system state is **OK**, if the current system configuration, that is the devices existing in the system and their FBlocks, are known to the NetworkMaster; otherwise the system state is **NotOK**. The system is in the NotOK state directly after the start-up or if an invalid configuration has occurred after a Network Change Event (e.g., due to an invalid device address).

In order to determine the system configuration during the start-up or after a Network Change Event, the NetworkMaster executes a system scan in which it queries the FBlocks of all devices via the function FBlockIDs. This query is sent with the physical device address.

If there is a valid configuration and the system status was NotOK, the system state changes to OK and the NetworkMaster notifies the devices in the MOST system by means of a broadcast of the function Configuration. The parameter Configuration-Control indicates the system state (in this case OK). Figure 4.9 shows the operational sequence of a system scan. In practice, the FBlocks also can be queried in parallel.

After the initial system scan, devices may also inform the NetworkMaster of new or disabled FBlocks by sending a status message of the function FBlockIDs with a new list of FBlocks. If the system state was OK before a system scan or the

NetworkMaster is informed of a change in the FBlocks offered by a device, the NetworkMaster distributes the information about any added or disabled FBlocks to the MOST system via the function Configuration. In the first case, the parameter ConfigurationControl has the value **NewExt**, in the second case, it has the value **Invalid**. With the additional parameter DeltaFBlockList, a list with FBlockIDs, InstIDs, and – in case of NewExt – the logical address, of the respective FBlocks is included.

If during a system scan a conflict occurs between the functional addresses of FBlocks, which means that several instances of an FBlock have the same InstID, the NetworkMaster tries to resolve the conflict by resetting the InstID that was reported last. If an error occurs, such as an invalid device address, which cannot be solved by resetting InstIDs, the NetworkMaster sets the system state to NotOK and thus triggers a new start-up of the system.

The information on the current configuration of the MOST system is stored by the NetworkMaster in a central data base – the Central Registry. The Central Registry stores the logical address of each device in the system, as well as a list of the implemented FBlocks with their InstIDs. The other devices in the system can query its current configuration via the function CentralRegistry and thus determine the device address associated with the functional address of a communication partner. If the higher levels of the Network Service are used, this mechanism is executed automatically.

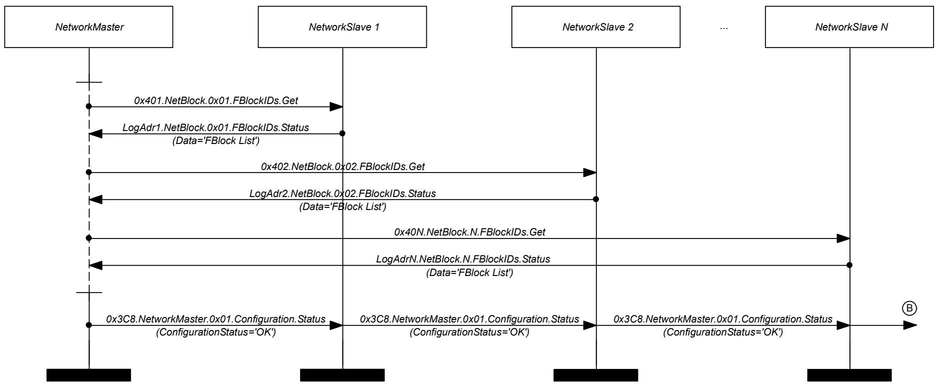

NetworkMaster
NetworkSlave 1
NetworkSlave 2
...
NetworkSlave N
0x401.NetBlock.0x01.FBlockIDs.Get
LogAdr1.NetBlock.0x01.FBlockIDs.Status
(Data='FBlock List')
0x402.NetBlock.0x02.FBlockIDs.Get
LogAdr2.NetBlock.0x02.FBlockIDs.Status
(Data='FBlock List')
0x40N.NetBlock.N.FBlockIDs.Get
LogAdrN.NetBlock.N.FBlockIDs.Status
(Data='FBlock List')
0x3C8.NetworkMaster.0x01.Configuration.Status
(ConfigurationStatus='OK')
0x3C8.NetworkMaster.0x01.Configuration.Status
(ConfigurationStatus='OK')
0x3C8.NetworkMaster.0x01.Configuration.Status
(ConfigurationStatus='OK')
B

Fig. 4.9: Operational sequence of a system scan by the NetworkMaster

The FBlock NetworkMaster implements the following functions:

- **Configuration** (FktID: 0xA00): With the function Configuration, the NetworkMaster disseminates information about the system state in a MOST system by means of a broadcast (the parameter ConfigurationControl has the values

OK or NotOK). Additionally, newly added or disabled FBlocks are announced (ConfigurationControl: NewExt or Invalid).

- **CentralRegistry** (FktID: 0xA01): Enables the querying of the current Central Registry. The complete Central Registry can be queried, all instances of a given FBlock can be retrieved, or a specific instance for an FBlock is queried. The function CentralRegistry returns a corresponding list of triplets of logical device address, FBlockID and InstID.
- **SystemAvail** (FktID: 0xA10): The corresponding optional mechanism provides information about the availability of the system. To this end the NetworkMaster queries the function ImplFBlockIDs of all devices in the system. With the parameters DeviceAvail and FBlockAvail a Slave device may determine if all devices have correctly responded to the NetworkMaster and if all implemented FBlocks are currently available.

FBlock ConnectionMaster (FBlockID: 0x03)

The FBlock ConnectionMaster is used for the administration of the streaming connections in the MOST system. It will be discussed in the next section.

FBlock PowerMaster (FBlockID: 0x04)

There is no particular FBlock specified for the PowerMaster. The FBlockID 0x04 is only used for indicating the device with the corresponding functionality.

The PowerMaster is responsible for coordinating the shutdown of a MOST system. This may be a regular shutdown, for example when the engine is switched off, or an emergency measure, for example when the temperature is too high.

To shut down the other devices in the MOST system, the PowerMaster uses the function ShutDown of the NetBlock. It first announces the shutdown by sending a Shutdown.StartAck(SenderHandle, Query) message as a broadcast. This enables the other devices to delay the shutdown by means of a ShutDown.ResultAck(SenderHandle, Suspend) message. Depending on the system architecture, an ongoing telephone call, for example, could prevent the shutdown.

The PowerMaster repeats the announcement of the shutdown at regular intervals. If it does not receive a ShutDown.ResultAck(SenderHandle, Suspend) message as a response, it initiates the actual procedure for shutting down by broadcasting the message Shudown.StartAck(SenderHandle, Execute).

If a device detects a high temperature, it can force the shutdown of the system by broadcasting a ShutDown.ResultAck(SenderHandle, TemperatureShutdown) message. The PowerMaster then executes the above described procedure for a normal shutdown of the system.

4.3.5 Administration of Streaming Connections

Due to the fact that the transmission of multimedia data is a central application area for a MOST system, the administration of the streaming connections, which are used to transmit audio and video data, plays a central role in a MOST system. The following section deals with the mechanisms and interfaces provided for that purpose in the application layer.

Overview

In the MOST Specification, sources and sinks for multimedia data (audio and video) are conceptually associated with an FBlock. Each FBlock can have one or more sources and – at the same time – one or more sinks. The telephone FBlock, for example, has a source for voice output via the loudspeaker system of the vehicle, as well as a sink for voice input via the vehicle's microphone system.

Each source of an FBlock is addressed by the unambiguous tag ***SourceNr*** of type Unsigned Byte. The sinks of an FBlock are identified correspondingly via the tag ***SinkNr***.

The function StreamDataInfo can be used to determine which sources and sinks are implemented by a specific FBlock:

- **StreamDataInfo** (FktID: 0x116): Returns a list with the source numbers and a list with the sink numbers of the FBlock.

For a source, a streaming connection can be created, which is characterized by its connection label and its bandwidth. The connection label is an unambiguous identifier and described by the parameter ***ConnectionLabel*** of type Unsigned Word. The bandwidth is described by the parameter ***BlockWidth*** also of type Unsigned Word, which defines how many bytes per MOST Frame are transmitted for this connection. One or more sinks can be attached to an existing connection.

Figure 4.10 shows the procedure used when establishing a streaming connection between the source of one FBlock and the sink of another FBlock. The respective functions for sources and sinks are discussed in the next paragraph.

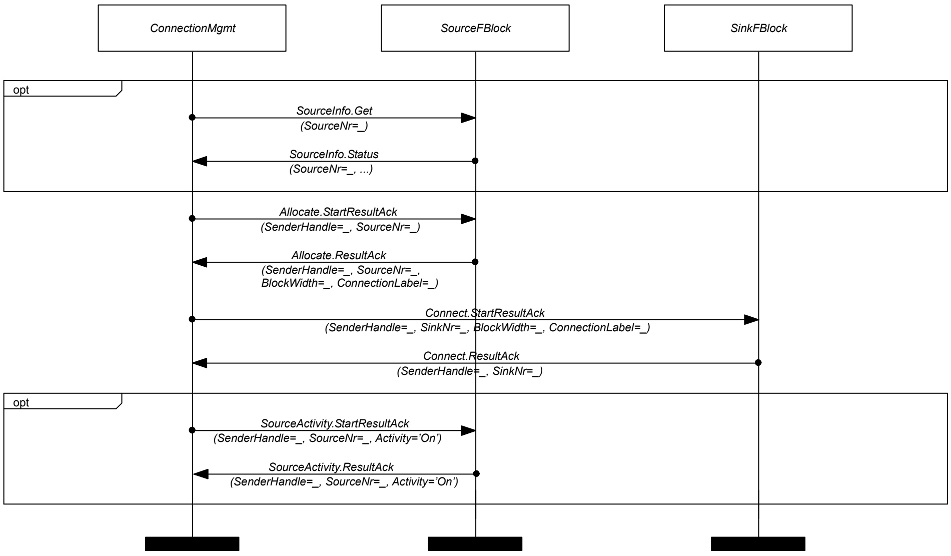

ConnectionMgmt
SourceFBlock
SinkFBlock
opt
SourceInfo.Get (SourceNr=_)
SourceInfo.Status (SourceNr=_, ...)
Allocate.StartResultAck (SenderHandle=_, SourceNr=_)
Allocate.ResultAck (SenderHandle=_, SourceNr=_, BlockWidth=_, ConnectionLabel=_)
Connect.StartResultAck (SenderHandle=_, SinkNr=_, BlockWidth=_, ConnectionLabel=_)
Connect.ResultAck (SenderHandle=_, SinkNr=_)
opt
SourceActivity.StartResultAck (SenderHandle=_, SourceNr=_, Activity='On')
SourceActivity.ResultAck (SenderHandle=_, SourceNr=_, Activity='On')

Fig.4.10: Establishment of a streaming connection

Functions for Sources

The MOST Specification requires the following functions for FBlocks with sources of multimedia data. These functions are specified in the specification of the General FBlock [GenFB] and are implemented by every FBlock with a source:

- **SourceInfo** (FktID: 0x100): The property SourceInfo returns information about the properties of the indicated source, such as its content type (e.g., PCM, SPDF or MPEG2).
- **SourceName** (FktID: 0x104): Returns a string that identifies the indicated source.
- **SourceActivity** (FktID: 0x103): A source can be set to active or inactive using the optional function SourceActivity. Accordingly, the source puts out data on the attached streaming connection.
- **Allocate** (FktID: 0x101): By means of the function Allocate, the FBlock is requested to create a streaming connection and to connect the indicated source with it. At a certain point of time, only one source can be connected to a specific streaming connection.
- **AllocateExt** (FktID: 0x108): Is similar to the function Allocate but contains a further parameter ClkSrcLabel to indicate the connection label for the clock source that is necessary for isochronous connections.
- **DeAllocate** (FktID: 0x102): Disconnects the indicated source from its streaming connection and deallocates it at the TimingMaster.

Functions for Sinks

The following functions for FBlocks having a sink for multimedia data are defined accordingly:

- **SinkInfo** (0x110): Similar to SourceInfo, this property contains information about the indicated sink.
- **SinkName** (0x114): The identification string of the sink.
- **Mute** (0x113): This function toggles the mute status of a connection. It also enables querying the corresponding current status of the sink, in case, for example, the status was set automatically by the FBlock to handle an error. The Mute function is optional, since more complex mechanisms for that purpose are used in some systems.
- **Connect** (0x111): Connects the indicated sink of the FBlock to the corresponding streaming connection. More than one sink can be connected to one streaming connection at one time.
- **DisConnect** (0x112): Disconnects the indicated sink from its streaming connection.

FBlock ConnectionMaster (FBlockID: 0x03)

The MOST Specification defines the ConnectionMaster as a central administrative unit for streaming connections. It establishes or removes streaming connections between given sources and sinks on request. Additionally, it administrates the information about all connections that exist in a system.

For this purpose, the FBlock ConnectionMaster implements the following functions (see [FBCM]):

- **BuildConnection** (FktID: 0x200): Establishes a connection between a source and sink given by the respective FBlockIDs and InstIDs as well as the SourceNr and SinkNr. If necessary, a streaming connection is reserved. Basically, the ConnectionMaster executes the procedure shown in figure 4.9.
- **RemoveConnection** (FktID: 0x201): Removes an existing connection between an indicated source and a sink. The corresponding streaming connection is released, if necessary.
- **ConnectionTable** (FktID: 0x400): Keeps a list of all streaming connections currently established in the system.
- **AvailableChannels** (FktID: 0.401): Indicates the remaining bandwidth for streaming connections in a MOST frame (in bytes).
- **MoveBoundary** (FktID: 0x402): This function can be used to adjust the boundary, that is the available bandwidth between streaming and packet connections in a MOST system.

- **BoundaryChange** (FktID: 0x403): A device can set a notification of this property to be informed about an onging boundary change.

When a central instance of an infotainment system administers the streaming connections throughout the system considering additional criteria, such as a priority of applications, this instance often also adopts the tasks of the ConnectionMaster. The FBlock ConnectionMaster is, in this case, often only implemented for debugging purposes.

4.4 Standardized Application Interfaces

Besides the system FBlocks the MOST Cooperation also standardizes FBlocks for the interfaces of typical applications in vehicle infotainment systems.

This chapter provides an overview of the specified application interfaces. Some selected interfaces will be discussed in more detail as examples. Finally, a more detailed example for using an application FBlock in a MOST system is given, taking as an example the AuxIn FBlock, which provides access to mobile MP3 players.

4.4.1 Function Library

The function catalog contains the current specifications for all FBlocks defined by the MOST Specification. It contains not only the system FBlocks described in section 4.3.4, but also the interfaces standardized for applications. Table 4.7 shows the current content of the function catalog with all FBlocks together with their most recent version number.

Kind	*Name*	*FBlockID*	*Version*[1]	*Description*
	GeneralFBlock	-	3.0.2	Common coordinated functions
	GeneralPlayer	-	2.5.1	Coordinated functions for player devices
Administration	NetBlock	0x01	3.0.1	System-FBlock implemented by every device
	NetworkMaster	0x02	3.0.1	NetworkMaster
	ConnectionMaster	0x03	3.0.1	ConnectionMaster

[1] As of 2nd quarter 2010.

Kind	*Name*	*FBlockID*	*Version*	*Description*
	Vehicle	0x05	1.6	Interface for vehicle-related data
	Diagnosis	0x06	3.0	Access to diagnostic functions
	DebugMessages	0x07	1.0.1	Functions for controlling debug output
	Tool	0x0E	1.0	FBlock for test purposes
	EnhancedTestability	0x0F	3.0.1	Necessary for compliance tests
Audio	AudioAmplifier	0x22	2.4.2	Amplifier
	AuxIn	0x24	3.5	Interface for mobile consumer electronic devices
	MicrophoneInput	0x26	2.3.1	Interface for microphones
Drives	AudioTapePlayer	0x30	2.3.1	Tape deck
	AudioDiskPlayer	0x31	2.4	CD player or CD changer
	DVDVideoPlayer	0x34	3.0	DVD player
Receiver	AmFmTuner	0x40	2.4.2	Radio tuner
	TMCTuner	0x41	2.3.1	Tuner for traffic messages (TMC)
	TVTuner	0x42	2.3.2	TV tuner
	DABTuner	0x43	3.0, 4.0	DAB tuner
	SatRadio	0x44	2.4	Tuner for Satellite Digital Audio Radio Service
Communication	Telephone	0x50	2.3.2	Interface for connecting a mobile telephone
	GeneralPhonebook	0x51	2.3.2	Phonebook of a telephone
	NavigationSystem[1]	0x52	1.11	Navigation unit
	GraphicDisplay	0x60	2.3	Display unit

[1] Only unofficially released by the MOST Cooperation.

Kind	*Name*	*FBlockID*	*Version*	*Description*
	UniqueFunctions	0xFF	2.3	Collection of functions for the area of unique functions

Table 4.7: FBlocks standardized by the MOST Cooperation

In table 2-2 of the MOST Specification [MOST 3.0], further FBlockIDs are defined (such as the FBlockID 0x54 for Bluetooth), that have no specifications in the function catalog. These are either specifications, which were not published officially, or they are placeholders for historic or future definitions.

4.4.2 Common Functions (GeneralFBlock)

Coordinated functions (0x000 to 0x1FF) must be implemented in the same way by all FBlocks. They are administered by the MOST Cooperation and described in the specification of the General FBlock, which is currently available in the version 3.0 [GenFB].

In addition to the functions mandatory for FBlocks (FktIDs, Notification and NotificationCheck), the GeneralFBlock describes functions for the administration of data structures, such as for inserting elements into an array (see section 4.3.2). The functions for the administration of streaming connections are also defined here (see section 4.3.5).

The GeneralFBlock only serves as a template. When defining an FBlock, the necessary functions from the current version of the GeneralFBlock are adopted as part of the FBlock specification (i.e., they are not influenced by changes in a possibly new version of the GeneralFBlock). If, however, an FBlock implements a coordinated function, this function must not deviate from the specification in the GeneralFBlock.

4.4.3 Application Interfaces

In order to provide an insight into the application interfaces standardized by the MOST function catalog, some of the interfaces for important applications from the area of vehicle infotainment system are described here in more detail. For lack of space, it is not possible to give a detailed description of all FBlocks of the function catalog.

CD Player, CD Changer (AudioDiskPlayer, FBlockID: 0x31)

The FBlock AudioDiskPlayer [FBADiskPl 2.4] describes a simple CD player or CD changer. Since the MOST Cooperation defines FBlocks for different playing devices (besides the one for a CD player, there is one for a cassette player), their general functions are summarized in the FBlock GeneralPlayer [FBGenPl 2.5]. The AudioDiskPlayer is a simple implementation of the GeneralPlayer. The latter also provides additional functions such as for playing DVDs.

The FBlock AudioDiskPlayer describes the following groups of important functions:

- The functions described in section 4.3.5 are implemented for accessing an audio source. Via this source, the audio data of the currently played track can be made available in the MOST system, for example for output on an amplifier.
- The current device status is contained in properties, such as *DeckStatus*, and can be read out or set via corresponding operations. DeckStatus enables the execution of the usual actions for a player device, such as play, pause, or stop. Specific modes of the device can be set through further functions such as random, scan, or repeat.
- Functions like *AudioDiskInfo* provide information about the inserted CD (or CDs), concerning the type of CD (audio, video or CDROM) and its format (e.g., CDDA, ISO9660). In the case of a CD changer, the status of the magazine (*MagazineStatus*) and the active CD (*ActiveDisk*) can be queried.
- Additional functions give information concerning the play operation such as the number of the current track (*TrackPosition*) or its play time (*TimePosition*). A Controller usually requests a notification for these functions, in order to indicate, for example, the remaining duration of a track on the user interface or to be informed when the player changes to the next track in order to update the meta information for this track. To some extent, the play operation or the selection of the current track via TrackPosition can be influenced through these functions.

Amplifier (AudioAmplifier, FBlockID: 0x22)

A device with an audio amplifier that receives audio data, processes, amplifies, and outputs the data via the loudspeaker system of the car implements the FBlock AudioAmplifier [FBAudioA 2.4]:

- The audio data is received over one or more sinks, for which the FBlock implements the necessary functions as described in section 4.3.5.
- Typical settings for the output of the audio signal (e.g., volume, bass, fader, and treble) are controlled via corresponding functions.

- Additionally, the FBlock provides further functions to perform more specific adjustments of the amplifier functions such as activating and defining an equalizer function (*EqualizerOnOff*, *EqualizerSettings*).

Radio Tuner (AmFmTuner, FBlockID: 0x40)

The FBlock AmFmTuner [FBAmFmT 2.4] for an AM/FM tuner is described here as an example for the FBlocks defined by the MOST Specification for different kinds of radio tuners (e.g., the FBlock DABTuner for the digitial radio standard DAB or the FBlock SDARS for a digital satellite radio).

The following main functions are defined by the specification of the FBlock AmFmTuner:

- The functions *ATFrequency*, *ATWaveband* and *ATStationInfo* deliver information about the currently set frequency, the wave band and the station received (e.g., its name or program type).
- A station scan is started with the function *ATSeek* where different settings are possible (e.g., the direction of the scan, up or down).
- The FBlock AmFmTuner supports the administration of several lists with station memories, though a simple tuner only needs to implement one station list. By means of the function *ATPresetSave*, the current frequency can be stored in a specific list. The functions *ATPresetList1* and *ATPresetList2* provide access to station lists with detailed information, the functions *ATPresetShortList1* to *ATPresetShortList8* to lists with limited station information.
- Further functions control the behavior of the AM/FM tuner when there are traffic reports. The function *TAInfo* indicates the name of the current traffic program. An active traffic report is signalized by the function *TAMessage*. It can be interrupted by *TAEscape*. With *TPSwitch* the functionality for receiving traffic reports is switched on or off.
- Additionally, the FBlock AmFmTuner has functions for different options to control the behavior of the tuner. The function *RDSSwitch*, for example, switches the use of the RDS signal on or off.

Connection of Mobile Devices (AuxIn, FBlockID: 0x24)

The FBlock AuxIn is also based on the FBlock GeneralPlayer (see [FBAuxIn 3.5]) and is implemented by a MOST device that acts as a gateway to connect mobile MP3 players and other storage devices with music data to a MOST system.

The FBlock AuxIn supports mobile devices with their own administration of the stored music data such as Apple's iPod, or devices on which the data is only stored in a generic file system such as a USB stick. In the specification of the FBlock AuxIn, the former are called database devices, the latter mass storage devices. Additionally, devices can also be connected via an analog audio input.

Additionally to the functions of the GeneralPlayer already described for the FBlock AudioDiskPlayer, there are the following extensions:

- Some functions are an adaptation of functions of the GeneralPlayer that became necessary due to the much higher number of tracks or different access modes. These are *AuxTrackPosition* as adaptation for TrackPosition, *AuxTrackInformation* as supplement of TrackInformation, and *AuxTimeInformation* as supplement of TimePosition.
- Additional functions provide information about the connected device. Its device class as well as its concrete type can be determined via the function *AuxDeviceInfo*. The function *DeviceBrowsingCapabilities* informs about the categories of music files that can be queried on a database device such as querying all tracks of an artist or all tracks of a specific genre.
- In order to query the possibly very comprehensive metadata for the music tracks stored on a device, the FBlock AuxIn defines the function *SelectAudioListInfo*, which supports querying functions comparable to database queries. Two filters indicate which tracks are to be considered during the query and which information about the tracks (e.g., title of the song, artist, and name of the album) is to be sent back. In addition, the sorting sequence and a specific section for the data to be transmitted can be indicated for optimizing the presentation on the display. *SelectCurrentAudioListInfo* is similar to SelectAudioListInfo. However, while SelectAudioListInfo considers all music files stored on the attached device, SelectCurrentAudioListInfo only relates to the currently selected track list.
- With the function *SelectAudioListFilter*, the current list of the tracks for playback can be selected by setting a filter similar to the one used for querying in SelectAudioListInfo. The currently active filter is read out via the function *CurrentAudioListFilter*.
- In version 3.5 of the AuxIn FBlock, the functions *SelectCoverArt* and *SelectCurrentCoverArt* have been added to request information about the cover art stored with the music files on the mobile device. The function *RetrieveCoverArt* can be used to retrieve the corresponding picture file.
- Various other settings can be determined via additional functions; for example, whether the metadata is to be transmitted via the Control Channel or the Packet Data Channel (*AsyncControlSwitch*).

4.4.4 Example of Use (AuxIn)

This section explains the use of an FBlock in a MOST system, based on the example of the FBlock AuxIn. For this purpose, important message sequences when accessing the FBlock are described. These sequences are based on the ones described in the specification of the FBlock AuxIn [FBAuxIn 3.5]. They are only an example; the individual tasks could be implemented in a different manner.

When initializing the system, the Controller sets the notifications it requires, for example, for updating the HMI representation (see fig. 4.11). In case of the FBlock AuxIn, this is, amongst others, the status of the AuxIn device (DeckStatus), the number of the currently active track (AuxTrackPosition and TrackInformation), and its play time (TimePosition). Afterwards, the Controller is informed whenever the device has moved to the next track and can update the information on the HMI accordingly.

If the user requests a functionality of the connected device, the Controller must transform this request into a corresponding command to the FBlock AuxIn. In the following example (fig. 4.12) the device is set into play mode via the property DeckStatus. The resulting changes are sent back through the previously notified properties. The Controller uses this information, as well as the metadata of the now playing track that it queries actively from the Slave, for updating its HMI representation.

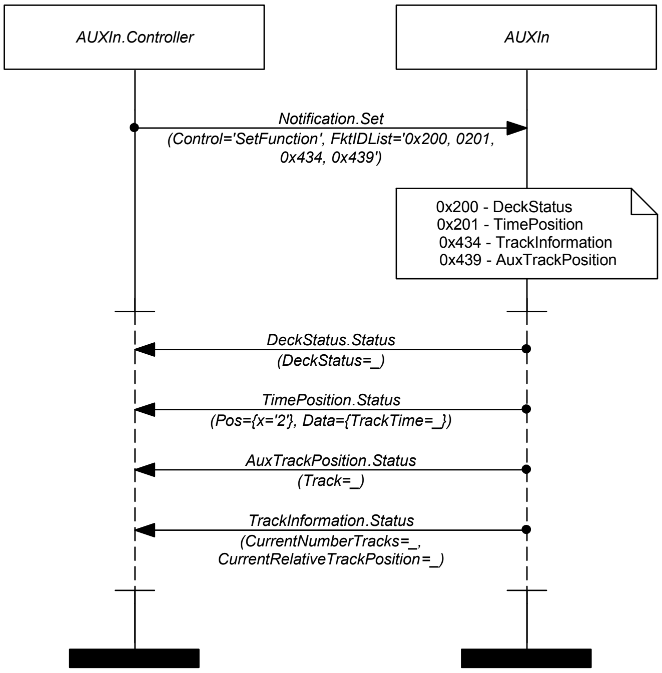


Fig.4.11: Setting of notifications during initialization

Figure 4.13 shows the querying of information about the currently playing music track. Via the function SelectCurrentAudioListInfo the identifier (tag), as well as the title, the artist, and the media type of the currently playing track is queried. The current track is indicated by the value zero for the parameter Start. In the response the actual position of the track in the current track list is contained; in our example it is the first track of the current track list.

The Controller initiates this sequence, for example, if it has requested an action of the FBlock AuxIn (see above) or if the Slave notifies a change of its status, for example that it has skipped to the next track.

If the user searches for a specific track by browsing through the media database of the device via the HMI of the Controller, the Controller has to query the information for the corresponding lists of media files from the FBlock AuxIn. Usually, the HMI offers different options for viewing the list of tracks, indicating specific properties of the tracks (artist, album, genre, songs).

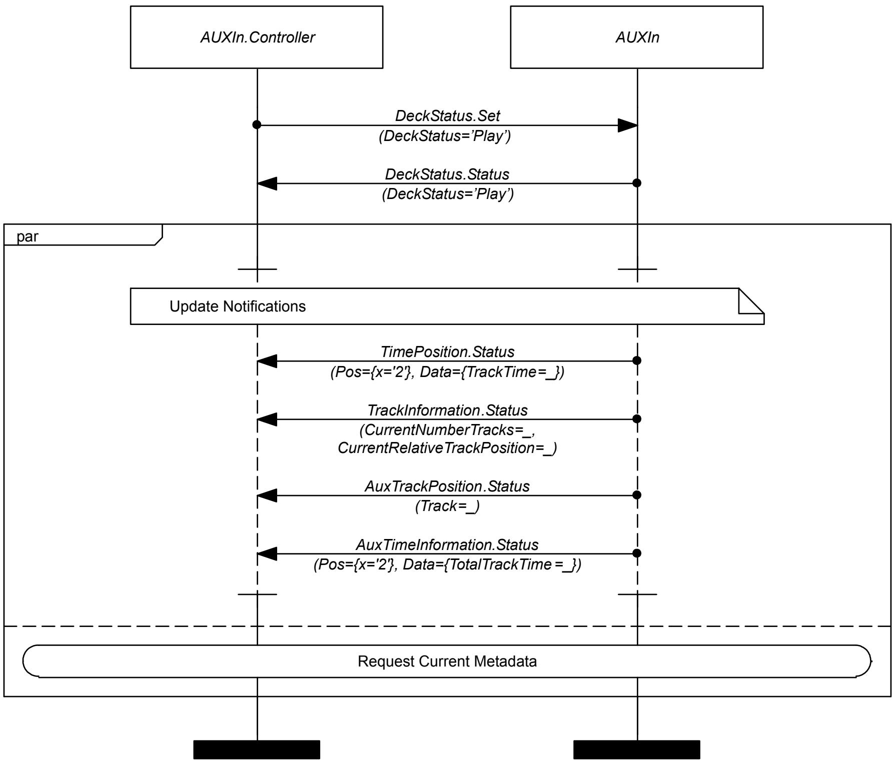


Fig.4.12: Setting the play mode

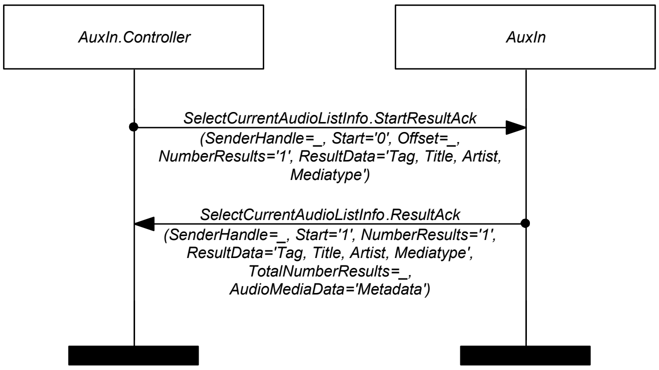

AuxIn.Controller
AuxIn
SelectCurrentAudioListInfo.StartResultAck (SenderHandle=_, Start='0', Offset=_, NumberResults='1', ResultData='Tag, Title, Artist, Mediatype')
SelectCurrentAudioListInfo.ResultAck (SenderHandle=_, Start='1', NumberResults='1', ResultData='Tag, Title, Artist, Mediatype', TotalNumberResults=_, AudioMediaData='Metadata')

Fig. 4.13: Querying of metadata for the currently playing track

In the example given in figure 4.14, it is assumed that the user is looking for a specific artist and that three entries can be shown on one page of the HMI. Information about the first three entries is thus queried via the function SelectAudioListInfo. The result starts with the first track of the selected track list, consisting of the tag, artist, and mediatype and is sorted by the names of the artists. If the user scrolls down in this list, the entries from the next page are queried by the Controller by increasing the starting point for the window by three. Usually the Controller builds up a cache of the following tracks in advance in order to speed up scrolling.

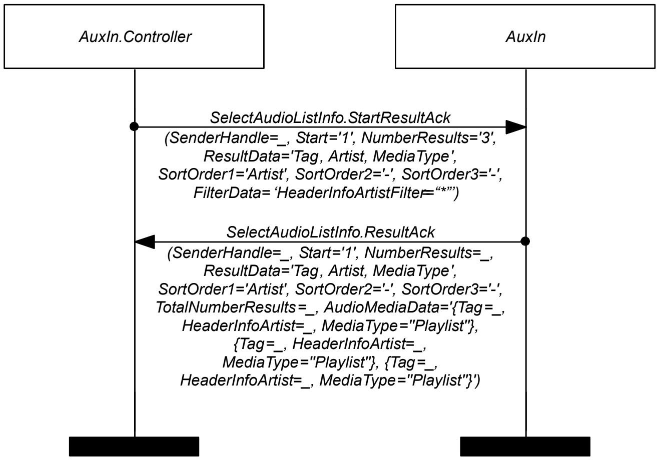

AuxIn.Controller
AuxIn
SelectAudioListInfo.StartResultAck (SenderHandle=_, Start='1', NumberResults='3', ResultData='Tag, Artist, MediaType', SortOrder1='Artist', SortOrder2='-', SortOrder3='-', FilterData= 'HeaderInfoArtistFilter="*"')
SelectAudioListInfo.ResultAck (SenderHandle=_, Start='1', NumberResults=_, ResultData='Tag, Artist, MediaType', SortOrder1='Artist', SortOrder2='-', SortOrder3='-', TotalNumberResults=_, AudioMediaData='{Tag=_, HeaderInfoArtist=_, MediaType="Playlist"}, {Tag=_, HeaderInfoArtist=_, MediaType="Playlist"}, {Tag=_, HeaderInfoArtist=_, MediaType="Playlist"}')

Fig. 4.14: Querying a list of artists

5 MOST Protocols

The MOST System uses either 44.1 kHz or 48 kHz sample rates for transmitting digital audio signals in hi-fi quality. Devices with a different sample rate can be adapted to the network by means of a sample rate conversion. In MOST systems, it is recommend to use 48 kHz.

Since the MOST System transmits the audio data synchronously, additional data buffering is not needed. This reduces the complexity of the device and thereby saves costs.

5.1 MOST Frame

The MOST system offers three frame types with different features: MOST25, MOST50, and MOST150.

5.1.1 MOST25 Frame

A MOST25 frame consists of 512 bits or 64 bytes. Sixty bytes are used for the transmission of stream and packet data. Two bytes transport one part of the control message that is made up of a total of 32 bytes for the administration of network and nodes. The control message is transported over 16 frames that are combined into one block. The first and the last byte both contain control information for the frame. Table 5.1 gives an overview.

Byte	*Bit*	*Description*
0	0-3	Preamble
	4-7	Boundary Descriptor
1	8-15	Data Byte 0
2	16-23	Data Byte 1
….	….	
60	480-487	Data Byte 59
61	488-495	Control frame Byte 0
62	496-503	Control frame Byte 1
63	504-510	Frame control and status bits
	511	Parity bit

Table 5.1: Structure of a MOST25 frame

Figure 5.1 shows the structure of a MOST25 frame.

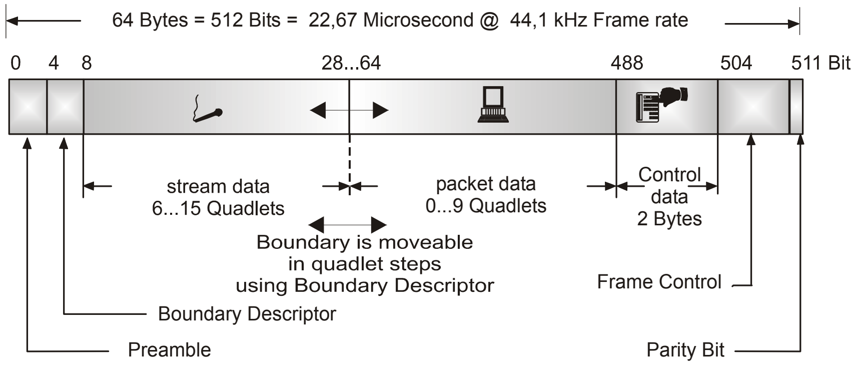


Fig. 5.1: MOST25 frame [MOST 2.4]

Preamble

The preamble of the MOST25 frame is used for synchronizing the TimingSlaves to the bit stream and for the initial identification of the frame. The slaves use a PLL switch to synchronize to the network.

The TimingMaster generates the preamble based on its oscillator frequency or its S/PDIF input signal.

After the bit stream has been carried over all MOST nodes, it arrives at the master with a phase shift caused by the signal propagation delay in each MOST node. The master thus synchronizes itself to the oncoming frame by means of its PLL, recovers all the bits, regenerates the frame and thus compensates the phase shift.

Boundary Descriptor

The Streaming Data Channel and the Packet Data Channel share a total of 60 bytes, which are available in a frame. The bandwidths of the two channels can be adapted to their corresponding requirements via the Boundary Descriptor.

The boundary between the two areas can be shifted in steps of 4 bytes (a quadlet). The Streaming Channel can thus have a width between 24 and 60 bytes (6 to 15 quadlets) and the Packet Data Channel a width between 0 and 36 bytes (0 to 9 quadlets).

The value of the Boundary Descriptor ranges from 6 to 15. It is administered by the MOST Network Interface Controller of the TimingMaster.

If the value of the Boundary Descriptor is changed by the TimingMaster, all synchronous connections have to be re-established.

The bandwidth of the Streaming and the Packet Channel is calculated by the following formula:

Packet bandwidth = Total bandwidth – (Boundary * 4) (Equation 5.1)

Streaming bandwidth = (Boundary * 4)

Packet bandwidth + Streaming bandwidth = 60

Frame Control

The last byte of the frame is used for controlling of the frame.

Parity Bit

The parity bit enables the detection of bit errors in the frame.

5.1.2 MOST50 Frame

The MOST50 generation uses a bit rate of 50 Mbit/s for doubling the bandwidth. The name MOST50 derives from this fact. Even though the sample rate does not change, the frame length can be increased to 1,024 bits or 128 bytes.

Figure 5.2 and table 5.2 show the structure of a MOST50 frame. The control data is placed in 4 bytes in the 11-bytes header, which also contains the Boundary Descriptor and the System Lock Flag. The bandwidth of the stream and the packet data can be adapted dynamically to current requirements without the need of re-establishing the synchronous connections. It is controlled by the property NetBlock.Boundary. The stream data can have a width of 0 to 29 quadlets plus one byte (0 to 117 bytes) and the packet data can also consist of 0 to 29 quadlets (0 to 116 bytes).

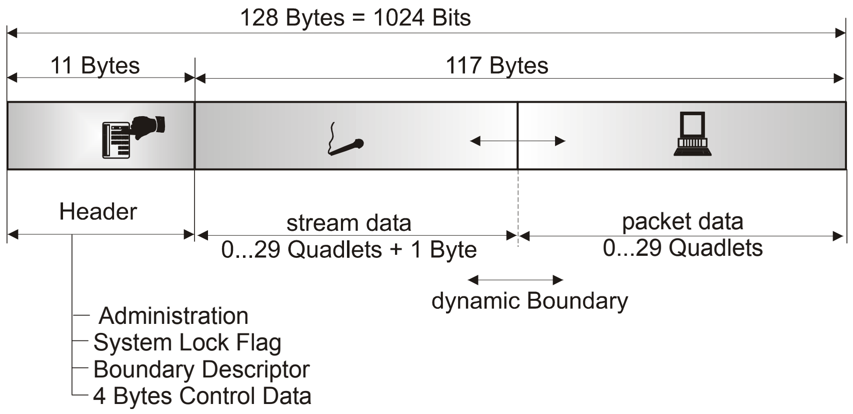


Fig. 5.2: MOST50 Frame [MOST 2.5]

The bandwidth of the Streaming and the Packet Channel is calculated by the following formula:

Packet bandwidth = Total bandwidth – (Boundary * 4) - 1

Streaming bandwidth = (Boundary * 4) + 1 (Equation 5.2)

Packet bandwidth + Streaming bandwidth = 117

Byte	*Bit*	*Description*
0-10	0-87	Administrative, includes System Lock Flag Boundary Descriptor 4 control data bytes
11-127	88-1023	data bytes

Table 5.2: Structure of a MOST50 frame

5.1.3 MOST150 Frame

The MOST150 frame has a similar structure to MOST50. Using the same sample rate und three times higher baud rate as MOST50, the frame length amounts to 3,072 bits or 384 bytes.

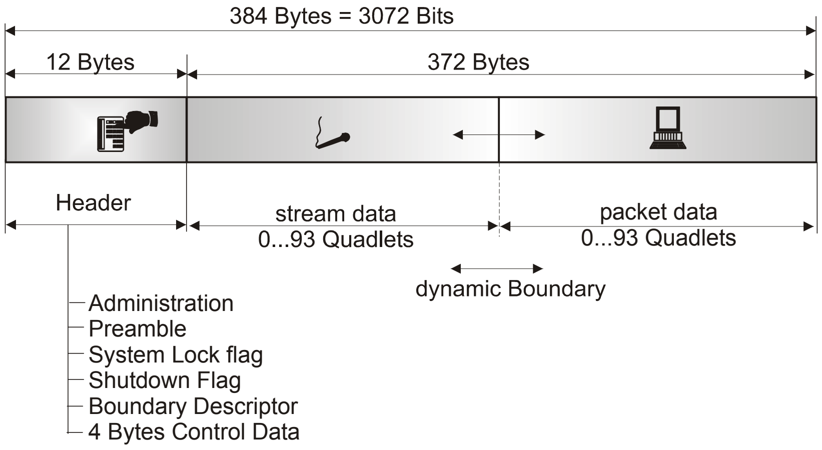


Fig. 5.3: MOST150 Frame [MOST 3.0]

Figure 5.3 and table 5.3 illustrate the structure of a MOST150 frame. The control data and the Boundary Descriptor are also placed in 4 bytes in the 12-bytes header. The bandwidth can be adapted dynamically too. The stream data can have a width of 0 to 93 quadlets (0 to 372 bytes) and the packet data 0 to 93 quadlets (0 to 372 bytes) as well. It is possible to use the total available bandwidth both for streaming data or for packet data.

Byte	*Bit*	*Description*
0-11	0-95	Administrative, includes • Preamble • System Lock Flag • Shutdown Flag • Boundary Descriptor • 4 control data bytes
11-383	96-3071	data bytes

Table 5.3: Structure of a MOST150 frame

Boundary Descriptor

The Boundary Descriptor is also controlled by the property NetBlock.Boundary and can be changed without the need of re-establish the synchronous connections. Only the application on the node with the TimingMaster function can change the value. The initial value is set in the TimingMaster's configuration.

The bandwidth of the Streaming Channel and the Packet Data Channel is calculated by the following formula:

Packet bandwidth = Total bandwidth – (Boundary * 4)

Streaming bandwidth = (Boundary * 4)

Packet bandwidth + Streaming bandwidth = 372

Table 5.4 summarizes the influence of the boundary value for all versions.

	MOST25		*MOST50*		*MOST150*	
	Available bandwidth (in number of bytes per frame)		*Available bandwidth (in number of bytes per frame)*		*Available bandwidth (in number of bytes per frame)*	
NetBlock. Boundary	*Stream-ing Data*	*Packet Data*	*Stream-ing Data*	*Packet Data*	*Stream-ing Data*	*Packet Data*
0			1	116	0	372
1			5	112	4	368
2	n/a		9	108	8	368
3			13	104	12	360
4			17	100	16	356
5			21	96	20	352
6	24	36	25	92	24	348
…						
14	56	4	57	60	56	316
15	60	0	61	56	60	312
16			65	52	64	308
…						
29			117	0	116	256
30	n/a				120	252
…			n/a			
92					368	4
93					372	0

Table 5.4: Boundary influence on the bandwidth for MOST25, 50, 150

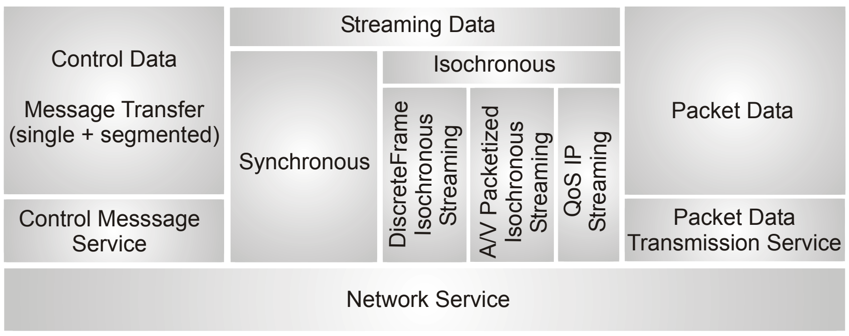

Control Data
Message Transfer (single + segmented)
Control Messsage Service
Streaming Data
Synchronous
Isochronous
DiscreteFrame Isochronous Streaming
A/V Packetized Isochronous Streaming
QoS IP Streaming
Packet Data
Packet Data Transmission Service
Network Service

Fig. 5.4: MOST data transport mechanisms

5.2 MOST Data Transport Mechanisms

MOST defines data transport mechanisms for control, streaming and packet data (see fig 5.4). The three MOST versions do not support all mechanisms. The following paragraphs describe them.

5.3 Streaming Data

The MOST Specification Rev. 3.0 distinguishes between synchronous and isochronous data. The latter can be transported only by MOST150.

The bandwidth of the Streaming Data Channel depends on the Boundary Descriptor (see section 5.1.2) and the frame version. It can be calculated using the following formula:

$$BW = SBC * 4 * 8Bits * Fs \text{ @MOST25; MOST150} \qquad \text{(Equation 5.3)}$$

$$BW = (SBC * 4 + 1) * 8Bits * Fs \text{ @MOST50}$$

BW:	Bandwidth
SBC:	Boundary Descriptor (synchronous bandwidth control)
Fs:	Sample rate

Summary

	MOST25	*MOST50*	*MOST150*
Sample rate (kHz)	44.1	48	48
minimum bandwidth (Mbit/s)	8.467	0.384	0
maximum bandwidth (Mbit/s)	21.168	44.928	142.848

Table 5.5: Bandwidth of the different frames

5.3.2 Synchronous Data

Synchronous data is used for the real-time transmission of audio and video data. Before the data can be transmitted a connection must be established by the ConnectionMaster, which uses the Application Message Service (AMS).

The NIC used a so-called Routing Engine (RE) for that purpose.

The INIC uses sockets instead of the Routing Engine. For this purpose, one socket has to be created and connected at the interfaces to the frame (MOST network port) and to the local resource (MediaLB or streaming port) (see also section 10.4.3).

Thus, up to 15 stereo connections (2x16 bits per channel) or 60 1-byte connections can be established simultaneously on a MOST25 frame. MOST50 can transport up to 29 stereo connections and MOST150 a maximum number of 93. Each connection has exactly one source and an arbitrary number of sinks. The content of the frame remains unmodified until the frame arrives back at the sending node.

The quasi-static establishment of connections on a channel is denominated as Time Division Multiplexing (TDM). The data are transmitted cyclically in a specified time pattern at the same frame position. There is no repetition in the case of communication errors. A valid value is then available during the next cycle.

5.3.3 Isochronous Data

MOST25/50 can solely transmit synchronous data. An MPEG data stream is artificially “stuffed” with bits, or the stream is converted (transcoded) to a fixed bit rate. Sample rate converters are sometimes used for audio applications to adapt PCM audio data to the MOST frame rate. However, not all audio or video data streams can be simply synchronized to this MOST time base.

MOST150 introduced isochronous transfer. Isochronous channels are handled in much the same way as synchronous channels in MOST: the required bandwidth is firmly reserved, which means channels for isochronous data are allocated, and the data is routed as needed. MOST150 can handle three different isochronous mechanisms designed for different applications. Accessing is provided by TDM and allocation of quasi-static physical channels.

A/V packetized isochronous streaming

This mechanism permits transfers of video data streams that arrive without any reference to the MOST time frame. The data is already consolidated in small data packets, which can optionally include a time stamp. MPEG data streams that are coded either with variable (VBR) or constant bit rates (CBR) are a typical example of this. A MOST transfer initially reserves the maximum isochronous bandwidth required. Figure 5.5 shows this mechanism.

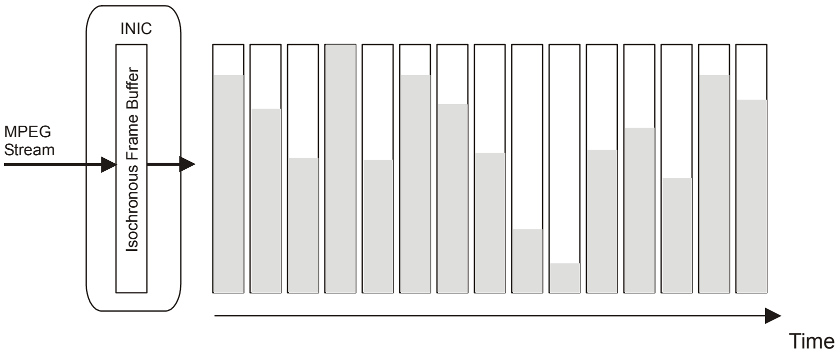


Fig. 5.5: A/V packetized isochronous streaming (source: SMSC Europa)

The data arriving at the MOST Network Interface Controller is transferred automatically by INIC to internal memory whose content is transferred cyclically via MOST. The Network Interface Controller of a data sink can copy the isochronous data from the bus to its internal memory, and pass it via a suitable interface on to the external hardware—an MPEG decoder, for example. The whole handling of MPEG data is encapsulated completely in the MOST150 INIC, so that all the application has to do is reserve the required isochronous bandwidth and configure the corresponding connections.

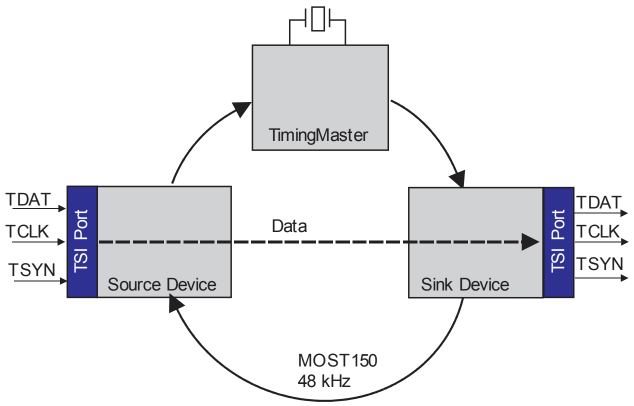


Fig. 5.6: Isochronous data transfer of MPEG data over an isochronous port/socket and the TSI (source. SMSC Europe)

Besides a range of standard interfaces, the MOST150 INIC includes a transport stream interface (TSI), now an established standard for video chipsets. Figure 5.6 shows the isochronous data transmission of MPEG data via one isochronous port/socket per node.

DiscreteFrame Isochronous streaming

Although audio data streams are synchronous in nature (a constant volume of data per time unit that is dispatched together with a highly accurate clock signal), they are often not synchronized to the same time base as MOST. One example is transfers of PCM audio data, which might have been sampled at a frequency of 44.1 (or 96) kHz, via a MOST150 network operating at a MOST frame rate of 48 kHz. The sample rate used for audio CDs is 44.1 kHz. Another example is the transfer of an S/PDIF signal (Sony Philips Digital Interface or Sony Philips Digital Interconnect Format) via MOST. S/PDIF is an interface standard used to interface audio data on decoder chipsets and in general used in the home audio domain, using optical TOSLINK or electric cinch and/or RCA jacks. Home audio equipment usually features components with one or more digital S/PDIF inputs or outputs. The interface is used for unidirectional transfer of PCM audio data according to IEC 60958 or compressed audio data according to IEC 61937 such as DTS, Dolby Digital (AC-3) or THX. Using isochronous transfer, these types of data streams can be "tunneled" via MOST, without the need to synchronize them to the MOST frame rate or convert the data with a sample rate converter.

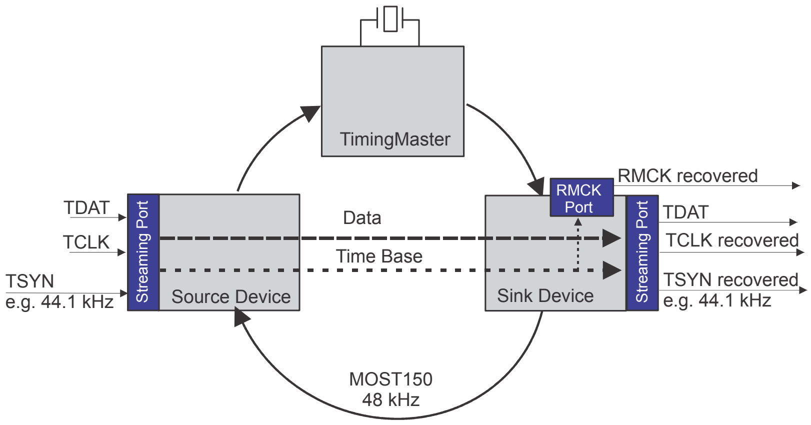


Fig. 5.7: Isochronous data transfer of audio data via two isochronous ports/sockets (source. SMSC Europe)

To transfer an audio signal isochronously, it is not enough to simply tunnel the data via MOST. In addition, the time base has to be maintained. This means the time base must be transferred and the sink needs to re-synthesize the clock with high precision.

The MOST150 INIC features inputs for non-MOST clock signals. The incoming signal needs to be sampled and the time base needs to be transferred along with the data. The time base (representing the clock) is restored in the sink after transfer through the network by a dedicated unit in the INIC, and the data is re-output at this clock rate.

Figure 5.7 shows the isochronous data transfer of audio data via two isochronous ports/sockets, one for data and the other for the time base.

QoS Isochronous mode

QoS Isochronous mode is also called QoS IP (Streaming) and is presented in section 5.8.3.

The MOST Stream Transmission Specification [MOST Stream] describes the isochronous data in details.

5.4 Packet Data Channel

The Packet Data Channel provides transmission of longer data packets and control data. Its bandwidth also depends on the Boundary Descriptor. It uses a data link layer protocol and transports packet data which is not transmitted cyclically (see section 5.7), such as the TCP/IP protocol or MOST High Protocol (MHP).

The MOST specification describes a total of four packet data protocols. Figure 5.6 shows the protocols for MOST25 and MOST 50, and figure 5.7 for MOST150.

5.4.1 Packet data protocol for MOST25 and MOST50

The protocols for MOST25 and MOST50 are used with two different data field lengths (fig. 5.8). The 48-byte data link layer protocol transports up to 48 data bytes and the other layer up to 1,014 data bytes.

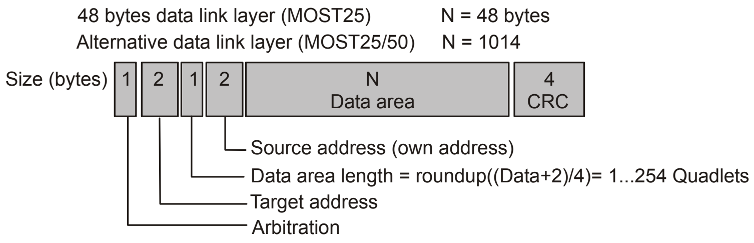

48 bytes data link layer (MOST25) N = 48 bytes
Alternative data link layer (MOST25/50) N = 1014
Size (bytes)
1
2
1
2
N Data area
4 CRC
Source address (own address)
Data area length = roundup((Data+2)/4)= 1...254 Quadlets
Target address
Arbitration

Fig. 5.8: Structure of the packet data protocols for MOST25 and MOST50

The NIC supports both data field lengths, but its internal cache is only sufficient for the 48-bytes protocol. The INIC uses the 48-bytes protocol, if it is controlled via the I²C bus. For the 1,014 byte data link layer protocol, it requires MediaLB (see also chapter 10).

The protocol header consists of the arbitration field (1 byte), in which the token is stored, the target address (2 bytes), the data field length (1 byte) and the source address (2 bytes). The protocol is secured by a CRC sum (4 bytes), which is automatically generated by the MOST Network Interface Controller. There is no automatic retransmission if there is a CRC sum error. This has to be handled by the higher layers.

5.4.2 Packet data protocol for MOST150

The packet data protocol of MOST150 has a different structure from MOST25/50 and provides two addressing mechanisms:

- 16 bit Target Address (classic MOST addressing)
- 48 bit Target Address (MAC addressing)

The CRC sum is placed in front of the data field and the data length is part of the header (fig. 5.9).

If the 16 bit addressing mode is used, the first 6 data bytes contain the header of the MOST High protocol.

The 48 bit addressing is used by Ethernet over MOST. In this case the CRC sum is calculated by the Ethernet FCS algorithm. See section 5.8.2 for more details.

Both addressing modes support low-level retries. The number of low-level retries and the propagation time between the retries is defined by the System Integrator.

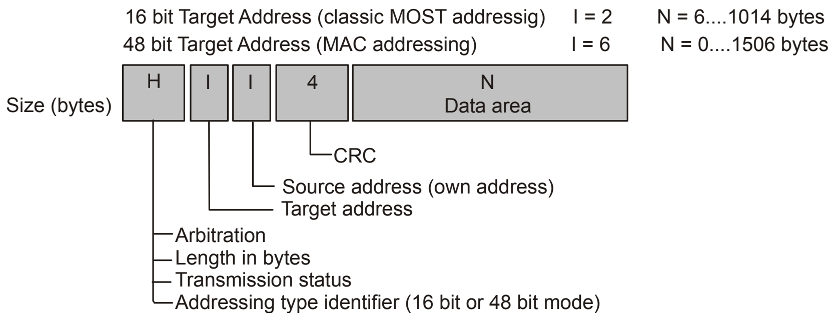

16 bit Target Address (classic MOST addressig) I = 2 N = 6....1014 bytes
48 bit Target Address (MAC addressing) I = 6 N = 0....1506 bytes
Size (bytes)
H
I
I
4
N
Data area
CRC
Source address (own address)
Target address
Arbitration
Length in bytes
Transmission status
Addressing type identifier (16 bit or 48 bit mode)

Fig. 5.9: Structure of the data link layer protocols for MOST150

5.4.3 Arbitration algorithm

Access control takes place on the packet data channel by means of a token (fig. 5.10). If none of the nodes wish to send data, the token is passed on from one node to the next.

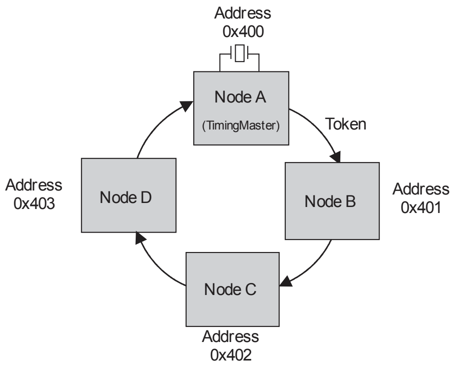


Fig. 5.10: Ring bus structure with four nodes

If a node wishes to send data, it has to wait for the token. It takes the token from the bus and thus gains the exclusive access authorization for the packet data channel. The node sends a single packet and subsequently places the token back onto the bus. If it has several packets to be sent, it has to wait for the token again.

The prioritization of the packet data transfer takes place via an access disclaimer of the token, i.e. a node with a low priority lets the token pass by several times – in spite of ready-to-transmit state - before taking it from the bus. This gives other nodes a higher priority for taking the token.

5.5 Control Channel

The Control Channel provides Control Message Services (CMS) that are used for network administration. It is generally specified for event-oriented transmissions at low bandwidth and short packet lengths. The Control Channel is secured by a CRC and has an ACK/NAK mechanism with automatic retry. The control messages provide single and segmented Application Message Services (AMS).

The following both sections introduce the different control messages in the three versions.

5.5.1 CMS for MOST25

Fig. 5.11 shows the structure of the control message for MOST25.

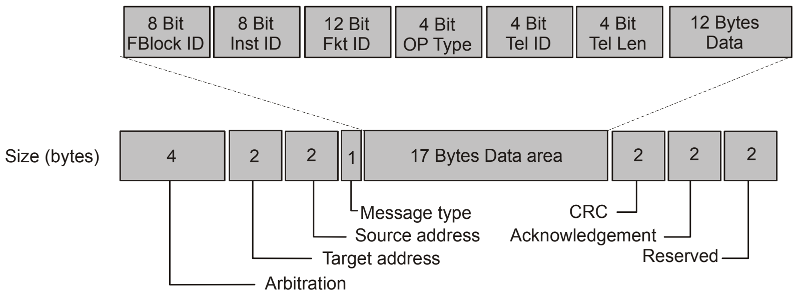


Fig. 5.11: Structure of a control message (source: [MOST 2.4]

In order to prevent the Control Channel from claiming too much bandwidth per frame, it is distributed over 16 frames which are combined into one block (see fig. 5.12). Each frame transports 2 bytes of the channel. The preamble of the first frame of a block has a specific bit pattern to identify the block.

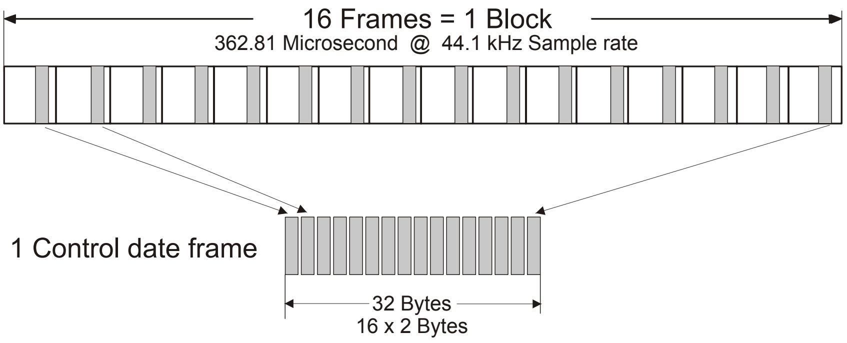


Fig. 5.12: Mapping of the Control Channel onto the MOST25 Frame

The protocol of the Control Channel has a constant length of 32 bytes.

Arbitration

The arbitration according to the Carrier Sense Multiple Access (CSMA) procedure relies on the first two frames (4 bytes). The access procedure guarantees fair bus allocation, even if the bus load is high, because the priority of a message is not dependent on the node position in the ring. The arbitration is executed automatically by the MOST Network Interface Controller.

Addresses

Bytes 4 and 5 contain the destination address, the two subsequent bytes contain the source address.

Message Type

Byte 8 transports one of the following message types (fig. 5.11):

- Normal message (0x00)
- System message
    - Resource Allocate (0x03)
    - Resource DeAllocate (0x04)
    - Remote GetSource (0x05)
    - Remote messages
        - Remote Read (0x01)
        - Remote Write (0x02)

Normal Messages

A normal message (message type: 0x00) contains data packets for the *Control Message Service* (CMS), i.e., it transports the data for modifying properties and starting functions of a function block, as well as the resulting response messages. A CD player, for example, can be controlled by means of a function block.

The actual data is transported in 17 data bytes. The normal messages can be sent to one receiver (single cast), to a defined group of receivers (group cast) or to all nodes (broadcast).

System Messages

The system messages have a code between 0x03 and 0x05. They are used exclusively by the MOST Network Interface Controller and are not visible to the application. The system messages for the resource management, such as Resource Allocate (message type: 0x03) and Resource DeAllocate (message type: 0x04) can be dispatched by all stream data sources. However, they are exclusively received and evaluated by the TimingMaster.

Remote functions, such as Remote Read (message type: 0x01) and Remote Write (message type: 0x02) are additional system messages. Through these functions, a NIC can write or read 1 to 8 data bytes into another interface controller. These functions are only intended for debugging and should not be used in the normal operating mode. The INIC no longer supports these message types, as it does not have a register wall (see chapter 10).

Message type 0x05 characterizes the Remote GetSource message. Through this message, the source of a streaming channel can be ascertained. It is normally sent as a broadcast message and ascertains the physical, the logical and the group address of the streaming source. If there are, by mistake, several sources for one channel, the message can be sent as single-cast message, in order to ascertain the sources and to resolve the conflict.

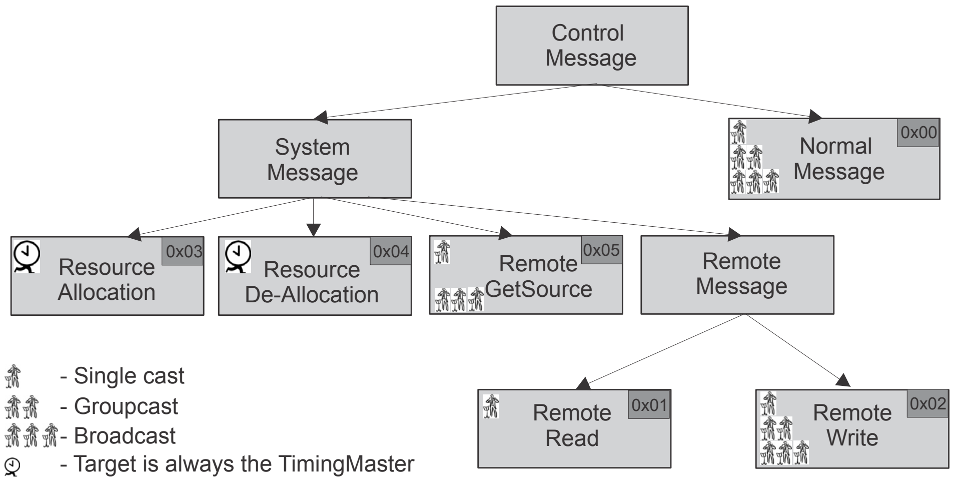


Fig. 5.13: Message types (source: [DS 8104])

CRC

The CRC sum enables the receiver to identify transmission errors.

Acknowledgement Flag

The receiver of a control message can inform the sender about error-free reception by using the acknowledgement flag (byte 28 and 29). The following states are possible:

- Message successfully received
- Cache blocked
- CRC error

The receiver can only write on the flag and has no possibility to reset it. Thus the sender can recognize, even in groupcast and broadcast messages that the message did not arrive properly in at least one of the receivers. It can, however, not discern which receiver is concerned.

By default, five repetitions are executed in case of communication errors. The number can be changed in the register bXRTY of the NIC (OS8104). For the INIC, the function *RetryParameters* is used. Before each repetition, the MOST Network Interface Controller (NIC) waits for a configurable number of blocks (11 by default). In the NIC (OS8104), the number can be set in the bXTIM (transmit retry time) register. The INIC, again, uses the function *RetryParameters*.

If it cannot successfully send the telegram by the last retry, it informs the host controller.

Data Field

The 17 data bytes transport the application message The structure, having a DeviceID, function block ID (FBlockID), instance ID, function ID, operation code and the related parameters, are very closely connected to the function blocks. The following syntax is used for the functional addressing on the application level:

```
DeviceID.FBlockID.InstID.FktID.OPType (Parameters)
```

Section 4.2 gives a detailed overview of that matter.

If a service needs more than 12 bytes for the parameters, the message must be segmented (TelID > 0). Additionally, the first data byte is used as telegram counter, i.e., there are only 11 data bytes available for the parameters. The first telegram has the TelID 1, the counter is set to zero. All further telegrams have the TelID 2. The counter continues from 0x01 to 0xFF and starts again with 0x00. The last telegram has the TelID 3. The TelLen indicates the number of bytes per telegram, i.e., for the TelID 1 or 2, the TelLen is always 12. It can be shorter for the TelID 0 or 3.

Data Rate

The number of control messages that can be transmitted each second depends on the sample rate. Only 62 of 64 transmitted blocks are available for control messages. Two blocks are used for the intra-system network administration such as the transmission of the channel resource allocation table. Since 16 frames are required for one control message, the number of messages per second is calculated according to the following formula [Found 04]:

$$CM = \frac{62}{64} * \frac{Fs}{16} = 2{,}670 Msg/s @ 44.1kHz \qquad \text{(Equation 5.4)}$$

with: CM: Control messages per second

Fs: Sample rate

Since each message carries 19 bytes (17 bytes data + 2 bytes DeviceID), the gross data rate (DR_{gross}) is calculated as follows:

$$\begin{aligned} DR_{gross} &= CM * 19 * 8 Bits \\ &= 405.84 kbits/s @ 44.1kHz \end{aligned} \qquad \text{(Equation 5.5)}$$

After the successful transmission of a control message, each node has to pause for two messages, i.e., the net data rate (DR_{net}) is one third of the gross data rate, provided that no higher prioritized node needs to send data:

$$DR_{net} = \frac{DR_{gross}}{3} = 135.28 kbits/s @ 44.1 kHz \qquad \text{(Equation 5.6)}$$

The real data rate also depends on the interface used at the NIC.

5.5.2 CMS for MOST50

MOST50 frames contain 4 bytes per frame for control data within the header. Therefore, a CMS message is split into 4 byte/frame pieces. In contrast to the MOST25 frame, the control message length can vary depending on the specifics of the message, hence a CMS is based on a variable number of frames. The result is better utilization of the bandwidth. The minimum number of single frames is 6 (TelLen = 0) for commands without payload and the maximum number of single frames is 9 (TelLen = 12). Figure 5.14 shows the control message structure. CMS messages with a payload greater than 12 bytes are transported by segmented transfer and have a maximum length of 65535 bytes. Like MOST25, it covers the AMS. The Message type no longer exits.

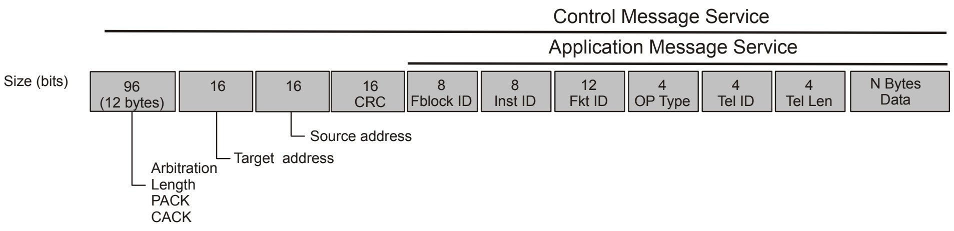


Fig. 5.14: Control message structure of MOST50

The header contains 12 bytes with arbitration, message length, preemptive acknowledge (PACK) and complete acknowledge (CACK).

Preemptive acknowledge (PACK)

The PACK byte is used for flow control. It is driven by the receiver and indicates either a receiver ready, full receive buffer or wrong target, leading to an aborted message.

Complete acknowledge (CACK)

A complete acknowledge (CACK) sent at the end of the message is also driven by the receiver and indicates a wrong CRC or successful transmission.

Data rate of a single frame

The number of messages per second is calculated according to the following formula:

$$CM = \frac{Fs}{number\_of\_frames} \qquad \text{(Equation 5.7)}$$

$$CM_{\max} = \frac{48000}{6} = 8{,}000 @ 48kHz; TelLen = 0$$

$$CM_{\min} = \frac{48000}{9} = 5{,}333 @ 48kHz; TelLen = 12$$

If none of the messages transport additional payload (TelLen = 0), 8,000 Control Messages per second is the best case. If all messages have the maximum size of payload (TelLen = 12) the result is 5,333 messages/s.

Payload$_{\min}$ amounts to 7 bytes (DeviceID, FBlockID, InstID, OPType, TelID, TelLen) and payload$_{\max}$ amounts to 19 bytes (payload$_{\min}$ + 12 bytes data). Therefore, the net rate is calculated with following formulas:

$$\begin{aligned} DR_{gross\_\min} &= CM_{\max} * payload_{\min} * 8Bits \\ &= 8{,}000 * 7 * 8Bits \\ &= 448kbit/s @ 48kHz; TelLen = 0 \end{aligned}$$

$$DR_{net\_\min} = \frac{DR_{gross\_\min}}{3} = 149.33kbit/s @ 48kHz; TelLen = 0$$

$$\begin{aligned} DR_{gross\_\max} &= CM_{\min} * payload_{\max} * 8Bits \\ &= 5{,}333 * 19 * 8Bits \\ &= 810.62kbit/s @ 48kHz; TelLen = 12 \end{aligned}$$

$$DR_{net\_\max} = \frac{DR_{gross\_\max}}{3} = 270.21 kbit/s@48kHz; TelLen = 12$$

MOST50 has a theoretical minimum net data rate of 149.33 kbit/s and a theoretical maximum net data rate of 270.21 kbit/s.

5.5.3 CMS for MOST150

MOST150 is similar to MOST50. The frames also contain 4 bytes per frame for control data within the header. The control message length also can vary depending on the specifics of the message. The minimum number of single frames is 6 (TelLen = 0) for commands without payload and the maximum number of single frames is 18 (TelLen = 45). Figure 5.15 shows the control message structure. CMS with the payload greater than 45 bytes are transported by segmented transfer and have a maximum length of 65535 bytes. MOST High protocol over the Control Channel is no longer supported.

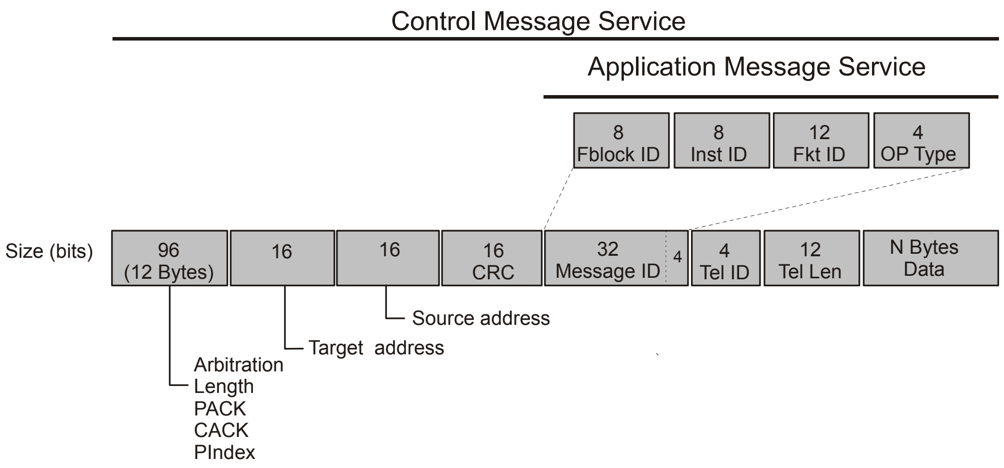


Fig. 5.15: Control message structure of MOST150 single frame

The header contains— additional compared to MOST50—the PIndex. The PIndex increments with each new control message sent by a particular node. It stays constant for low-level retries.

The Message ID covers the AMS parameters FBlockID, InstID, FktID and OPType. AMS sets the last 4 bits of the Message ID to 0x0F in the case of a segmentation error.

Data rate of a single frame

The number of messages per second is calculated according to the following formula:

$$CM = \frac{Fs}{number\_of\_frames} \qquad \text{(Equation 5.8)}$$

$$CM_{\max} = \frac{48000}{6} = 8{,}000 @ 48kHz; TelLen = 0$$

$$CM_{\min} = \frac{48000}{18} = 2{,}666 @ 48kHz; TelLen = 45$$

If none of the messages transport additional payload (TelLen = 0), 8,000 Control Messages per second is the best case. If all messages have the maximum size of payload (TelLen = 45) the result is 2,666 messages/s.

Payload$_{\min}$ amounts to 8 bytes (6 bytes Message ID, TelID, TelLen and 2 bytes target address) and payload$_{\max}$ amounts to 53 bytes (payload$_{\min}$ + 45 bytes data). Therefore, the net rate is calculated with following formulas:

$$\begin{aligned} DR_{gross\_\min} &= CM_{\max} * payload_{\min} * 8Bits \\ &= 8{,}000 * 8 * 8Bits \\ &= 512kbit/s @ 48kHz; TelLen = 0 \end{aligned}$$

$$\begin{aligned} DR_{gross\_\max} &= CM_{\min} * payload_{\max} * 8Bits \\ &= 2{,}666 * 53 * 8Bits \\ &= 1{,}130kbit/s @ 48kHz; TelLen = 45 \end{aligned}$$

MOST150 has theoretical minimum net data rate of 512 kbit/s and a theoretical maximum net data rate of 1,130 kbit/s.

Note that a MOST device cannot use the overall available control message bandwidth of the MOST network, typically caused by limitations by the IO interface, the device driver, the network stack and the application.

5.6 Summary

	MOST25	MOST50	MOST150
Frame size (bits)	512	1024	3072
Sample rate (kHz)	44.1	48	48
Streaming data			
Minimum (bytes)	24	0	0
Maximum (bytes)	60	117	372
minimum bandwidth (Mbit/s)	8.467	0.384	0
maximum bandwidth (Mbit/s)	21.168	44.928	142.848
Packet data			
Minimum (bytes)	0	0	0
Maximum (bytes)	36	116	372
minimum bandwidth (Mbit/s)	0	0	0
maximum bandwidth (Mbit/s)	10.841[1]	44.544	142.848
Control data			
Bytes per frame	2	4	4
Min. number of frames	16	6	6
Max. number of frames	16	9	18
Min. data bytes	19	7	8
Max. data bytes	19	19	53
Min. gross bandwidth (kbit/s)	405.84	448	512
Max. gross bandwidth (kbit/s)	405.84	810.62	1130

Table 5.6: Summary of the frame parameters of MOST25, 50 and 150

[1] for calculation see appendix B

5.7 Addressing

On level 2 of the ISO layer, the MOST network supports five modes of addressing for the data link layer protocol and the Control Channel:

- Internal Node Communication address (reserved for internal communication in a node)
- Node position address (RxTxPos)
- Logical node address (RxTxLog)
- Group address
- Broadcast address
- Ethernet MAC address (only MOST150)

16 bit addresses are always used, except for the Ethernet MAC address (it is 6 bytes long). Table 5.7 shows the division of the address domains. The four uppermost address bits are currently not in use; they are, however, reserved for future applications.

Address range	*Mode*
0x0000...0x000F	Internal Communication
0x0010…0x00FF	Static address range
0x0100…0x013F	Dynamic calculated (0x0100+POS) address range
0x0140…0x02FF	Logic address range; statically assigned
0x0300…0x03C7	Group addresses
0x03C8	Blocking broadcast address
0x03C9…0x03FE	Group addresses
0x03FF	Unblocking broadcast address
0x0400…0x043F	Node position (0x0400 + POS) address range
0x0440....0x04FF	Reserved
0x0500…0x0FEF	Logic address range; statically assigned
0x0FF0	Optional debug address
0x0FF1…0x0FFD	Reserved
0x0FFE	Init address of Network Service
0x0FFF	Init address of MOST Network Interface Controller
0x1000…0xFFFE	Reserved for future use
0xFFFF	Uninitialized logical node address

Table 5.7: Address ranges of the MOST network [MOST 3.0]

Node Position Address (RxTxPos)

Each node of a MOST network has a unique identification which depends on its position in the ring and is dynamically ascertained by the link layer. This identification is called node position. The TimingMaster always has node position 0x00, the following node 0x01, etc. (fig. 5.16). The maximum number of nodes in the ring is 64.

By means of the node position, the delay between signal source and signal sink can be determined quite easily by the source informing the sink of its position and the sink determining the number of network nodes through which the signal has passed.

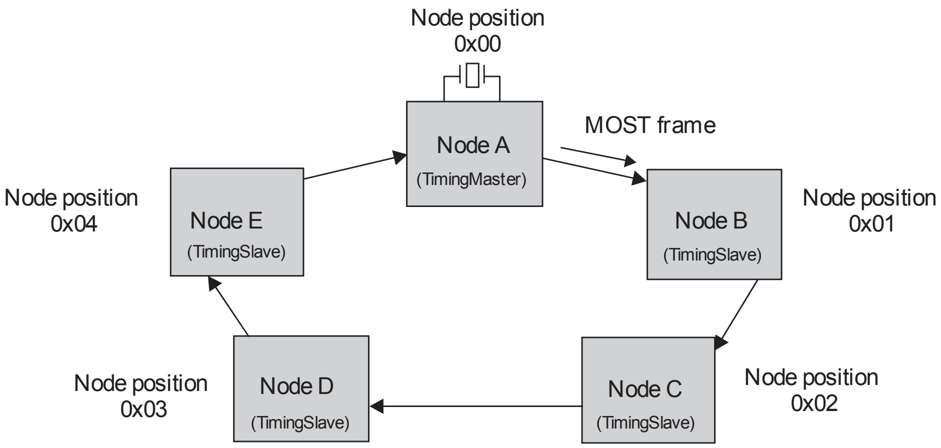


Fig. 5.16: MOST ring with the node addresses

The node position address is derived from the node position and is thus unambiguous in the network. When the system is started, the address is defined by the TimingMaster, as soon as the MOST Network Interface Controller is in lock status and has opened the bypass. The address domain starts with the address 0x400, which is always occupied by the TimingMaster. The node position address (RxTxPos) is calculated according to the following equation:

RxTxPos = 0x400+Pos

Since there is a possible maximum of 64 nodes in the ring, the last address is 0x043F; the remaining addresses are currently not being used.

Logical Node Address (RxTxLog)

The logical node address (RxTxLog) is a unique address in the network. After power-up, all uninitialized nodes have the address 0xFFFF. After the initialization, the logical node address is dynamically assigned according to the following equation (similar to the node position addresses) upon initialization of the network:

RxTxLog = 0x100+Pos

The TimingMaster has the logical address 0x100, the first node has the address 0x101, etc.

Anyhow, the system designer can statically assign the logical address in the address domains 0x140…0x2FF and 0x500…0xFEF. The address can already point at the function of the nodes (e.g., first video monitor 0x200, second video monitor 0x201), or at the position in the vehicle (e.g., 0x100 passenger cell left, 0x200 passenger cell right, 0x500 trunk).

In the NIC, the logical address is stored in two registers (bNAH and bNAL). The INIC uses the function *NodeAddress*. Before the NIC accepts the address, it can automatically check the uniqueness of the allocated address by means of the start-address-initialization function.

Group Address (GA)

The group address represents the allocation to a system class, such as amplifier, audio CD player or digital sound processor.

Group addresses occupy the address range 0x300…0x3C7 and 0x3C9…0x3FE. The initial group address is derived from the function block ID according to the following equation:

GroupAddress = 0x300+FBlockID

The FBlockID of the characteristic FBlock of the node is used here, assuming that this will be the most requested.

In the NIC, the group address is stored in the bGA (Group Address) register. The INIC uses the function *GroupAddress* for modification.

Broadcast

A blocking broadcast address is a unique address (0x3C8) addressing all nodes in the ring. All other control messages are delayed until all nodes have acknowledged the received broadcast. This reduces the bandwidth if used excessively.

An unblocking broadcast message (0x03FF) does not block the transmission of other control messages. This kind of transmission can be used for uncritical data transmission, which might not necessarily be received by all devices.

Due to the fact that broadcast messages require extensively utilize system resources, they should only be used for management functions, e.g., in order to receive the corresponding function blocks from all nodes during initialization. A further application area is querying all hardware and software versions of the components used in a car.

When a group address is used, the maximum data length of broadcast messages is reduced by one byte for the checksum.

Ethernet MAC Address

The Ethernet MAC address is a unique address in EUI-48 format and has a length of 48 bits (6 bytes) and is only supported within MOST150 networks. It is in the responsibility of each manufacturer to acquire its own OUI resp. block to build EUI-48 and MAC addresses.

The address supports the Ethernet-typical address comparison modes:

- Unicast (Perfect Filtering, 48 bit match)
- Multicast (Hash Filtering)
- Broadcast (0xFFFF FFFF FFFF)
- Promiscuous (unfiltered)

The mode is configured in the MOST Network Interface Controller.

5.8 Higher Protocols

The MOST Specification defines the *MOST High Protocol* and the adaptation layer MAMAC (MOST Asynchronous Medium Access Control) for the connection-oriented transmission of packet data on the transport layer level of the OSI model. This enables the TCP/IP stack to be attached to the data link layer protocol. By means of these protocols, packet data is sent with individual frames or as segments, for example navigation data. MOST150 networks do no longer support MAMAC because MOST150 offers a real Ethernet transmission.

5.8.1 MOST High Protocol (MHP)

The *MOST High Protocol* is a connection-oriented protocol and uses some of the mechanisms of the TCP protocol. However, the overhead for the control of the data flow was reduced to a minimum. As it is shown in figure 5.17, the protocol is attached to the Network Services Layer I. It can be implemented on the Packet Data Channel and is unidirectional, i.e. there is a transmitter and a receiver module. If data is to be transmitted bidirectionally, two connections are necessary.

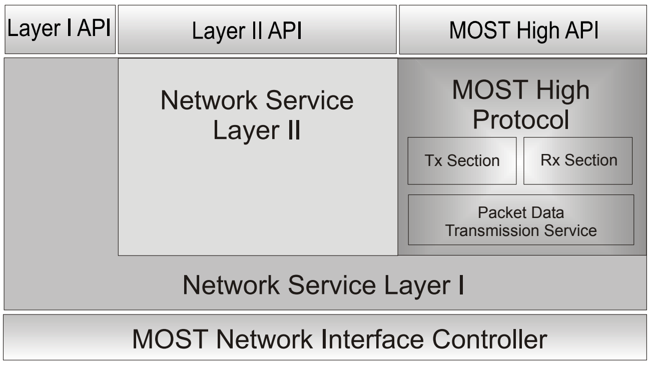


Fig. 5.17: Typical layer structure of the MOST High Protocol (source: [PFLNet])

Application Viewpoint

From the point of view of the application, the *MOST High Protocol* is located between Network Services and the function blocks of a MOST device (see fig. 5.18). The Controller sends application data to a function of a function block in the receiver. The interface between the Controller and the *MOST High Protocol* corresponds to the one for the control messages of the function blocks.

```
DeviceID.FBlockID.InstID.FktID.OPType(Data)
```

Since the protocol is a connection-oriented one, a communication always consists of three parts:

- Connection establishment
- Data transmission
- Connection termination

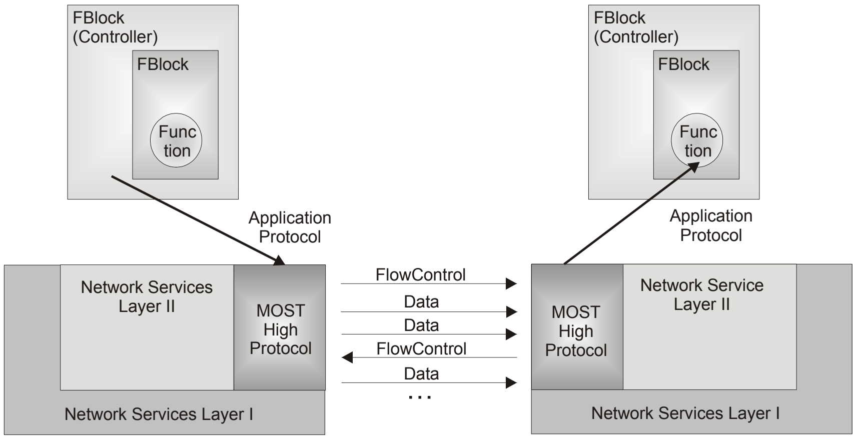


Fig. 5.18: View of the application to the MOST High Protocol (according to [MHP])

Additionally, the receiver informs the transmitter about the correct reception of the data by sending a positive acknowledgement, or requests the data again with of a negative acknowledgement. This is also characterized as flow control (FlowControl) (fig. 5.18).

	MHP on Control Channel	*MHP on Packet Data Channel*		
	(legacy only)	48-bytes data link layer protocol	1,014-bytes data link layer protocol	1,512-bytes data link layer protocol
Usable data bytes	17	48	1014	1524
Header length (bytes)	5	6	6	6
Flow control (bytes)	1	2	2	2
Max. data field length per frame (bytes)	11	40	1006	1518
Max. user frames per block	15	255	255	255
Bytes per block	1...165	1....10200	1...65535	1…65535

Table 5.8: MHP parameter for the control channel and for the transmission in the Packet Data Channel

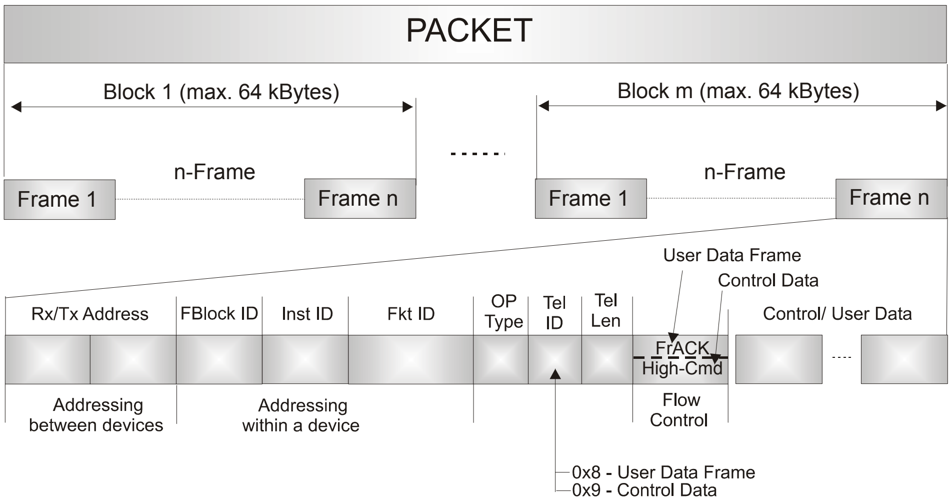


Fig.5.19: Structure of the MOST High Protocol (source: [MHP])

Protocol

MHP differentiates between packet, blocks and frames (see fig. 5.19). A packet consists of a number of blocks, which, in turn, consist of up to 256 frames. The number of frames in a block and the maximum data field length of the frames depend on whether MHP was implemented on the Control Channel (legacy only) or on the Packet Data Channel (table 5.8). The block size is always limited to 64 Kbytes.

High-Cmd	*Name*	*Description*
CA	Request Connection	Indicates the existence of data, the priority of connection, the maximum number of usable data bytes per frame, and the revision number of MHP, which is provided by the sender
F2	Start Connection	Command for establishing communication
F3	END Connection from sender	Connection termination request by sender
FC	END Connection from receiver	Connection termination request by receiver
FA	Frame ACK	Acknowledge of the receiver about received frame or block
FB	Request	Request for a certain single frame or an entire block
FF	Multiple Frames Request	Request for several frames of the current block
F0	Adjust Rate	Request from the receiver to increase or decrease the transmission rate
FD	Ready for Data	Acknowledge for connection is established
F1	Hold Connection from Sender	Indicates the interruption/continuation of the communication by the sender, e.g., if the transmit buffer is empty
FE	Hold Connection from Receiver	Indicates the interruption/continuation of the communication by the receiver, e.g., if the receiver buffer is full

Table 5.9: Commands of the MHP flow control (source: [MHIGH])

A frame consists of a Data Link Layer Header, which is structured analogously to the header of a control message, and the Transport Layer Header. For a flow control message (control data) the Transport Layer Header consists of the Command Token (High-Cmd). For a Data Frame (User Data Frame) it consists of the Frame Acknowledgement (FrACK). Figure 5.19 shows the frame structure.

Table 5.9 gives an overview of the commands of the flow control.

An acknowledgment can be sent for each frame of each block. Figure 5.20 shows a sequence diagram for MHP, in which an acknowledge is sent after each block. The individual steps can be understood on the basis of table 5.9. The 0-FRAME is the first frame of a block.

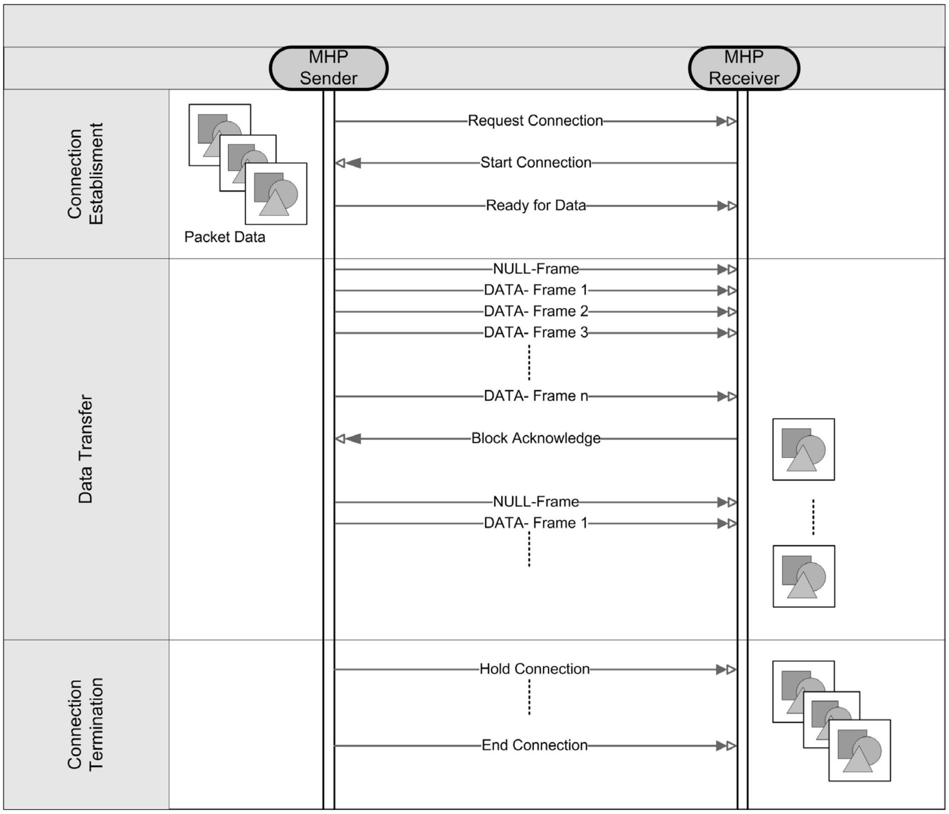


Fig. 5.20: MHP control and data flow

Additional information and the MSCs relating to the individual commands can be found in [MHIGH].

5.8.2 Ethernet over MOST

MOST Ethernet Packets

The MOST Packet Data Channel (PDC) is used for sending asynchronous packet data and supports two different addressing formats:

- 16 bit addressing; MOST Data Packets (MDP)
- 48 bit addressing; MOST Ethernet Packets (MEP)

MDP and MEP packets can be transmitted simultaneously by every device in the MOST network by sharing the bandwidth of the Packet Data Channel between these packet types. The arbitration mechanism guarantees fair access to the channel for every device. The bandwidth of the PDC can be dynamically changed by adjusting the Boundary Descriptor. The protocol is CRC protected and a packet is automatically retried if a CRC fail is detected by the receiver, or the receive buffers are full.

The MEP format supports 48 bit MAC addressing and is compliant with the IEEE Ethernet standard (refer table 5.10).

MAC DestAddr	*MAC SrcAddr*	*Data*	*CRC*
48 bits	48 bits	Maximum 1506 bytes	32bits

Table 5.10: Structure of the 48 bit MAC address

The data field can include an optional 32 bits VLAN (IEEE 802.1Q) field. However, the 16 bits length/type field is mandatory.

The INIC stores the MAC address and carries out address filtering, i.e., the filter can be set up to only pass packets to the EHC that are of interest to the application running on the EHC. The 48 bit addressing format supports the following standard IEEE Ethernet MAC address filter modes:

- Unicast (Perfect Filtering, 48 bit match)
- Multicast (Hash Filtering)
- Broadcast (0xFFFF FFFF FFFF)
- Promiscuous (unfiltered)

MEP packets transmit standard IEEE Ethernet frames on the Packet Data Channel. Therefore, a standard TCP/IP stack can be used without any change on top of the MOST Ethernet Driver.

Note: A low-level filter mechanism must be implemented to move MEP data to the MOST Ethernet Driver and MDP data to the standard MOST Network Services framework. The filter is necessary because MDP and MEP packets share the same data channel. Figure 5.21 shows the details.

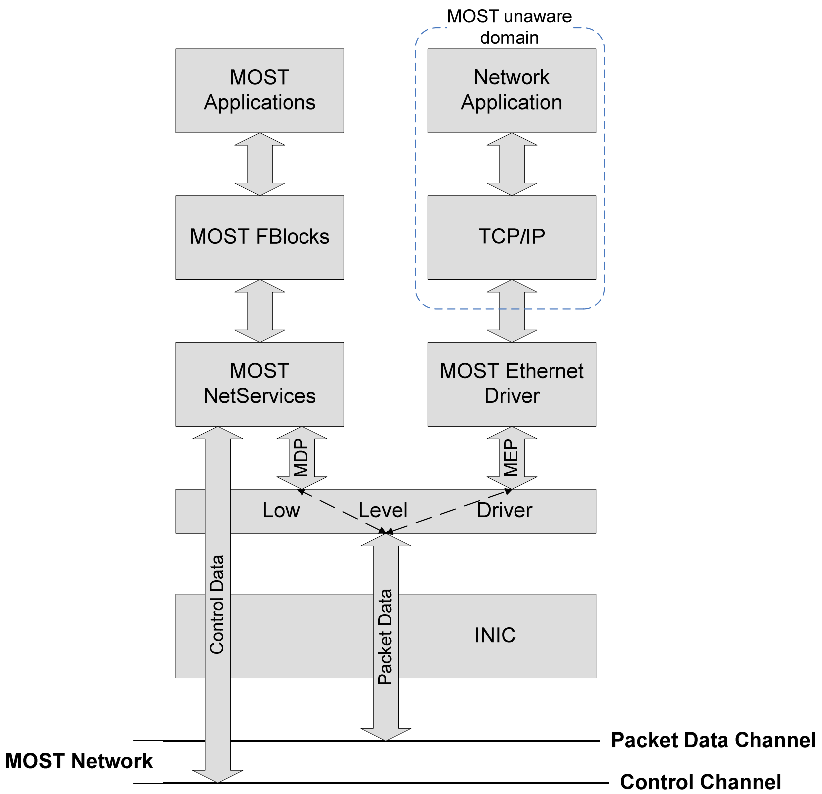


Fig. 5.21: MOST Ethernet driver concept

5.8.3 MOST Isochronous QoS IP (Streaming)

The MOST isochronous transmission subclass QoS IP (Streaming) can be used for transmitting packet data that requires guaranteed bandwidth, e.g., camera or video server applications.

The MOST Packet Data Channel shares the bandwidth between all devices on the MOST network. Therefore, it does not guarantee a certain bandwidth for an application. The QoS IP channel guarantees exclusive access to a certain bandwidth on the MOST network for a source device. It supports both point-to-point and point-to-multipoint connections. Furthermore, it is unidirectional and connection oriented. The connection is usually set up by a Connection Manager that instructs the source to allocate bandwidth on the MOST network and then instructs the sink(s) to connect to this particular channel.

Native MOST-aware applications use the QoS IP channels directly because they are aware of the QoS IP channels.

5.8.4 MOST Ethernet with QoS IP

The MOST Ethernet driver can handle MEP packets on the Packet Data Channel as well as QoS IP channels. It sets up single or multiple QoS IP channels for sending QoS classified packets to all targets. All other packets, which are not QoS classified, are sent on the shared Packet Data Channel. With this approach, a standard TCP/IP stack, IP-based communication protocols and network application can run on top of the MOST Ethernet driver without any adaption.

If QoS IP is required, an application only tags the TOS field in the IP header of the outgoing data, or it uses reserved ports for access to the QoS IP channels. Non QoS tagged packets will always be sent over the shared Packet Data Channel. Figure 5.22 shows the mechanism.

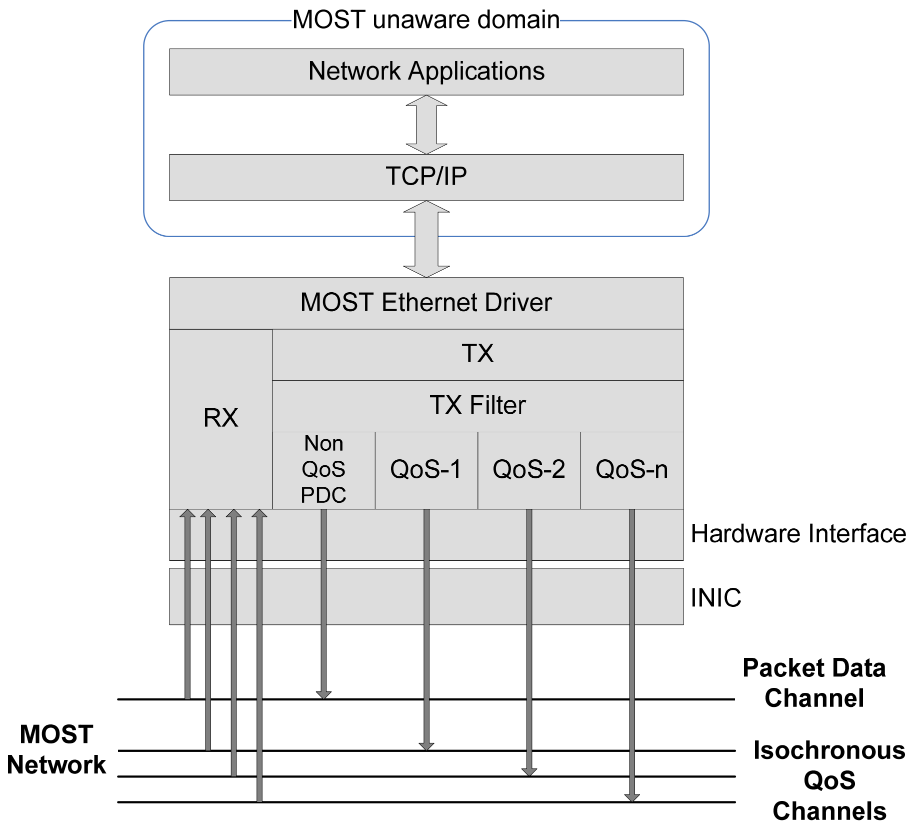


Fig. 5.22: MOST Ethernet with QoS

5.8.5 MAMAC

MAMAC (MOST Asynchronous Medium Access Control) is an adaptation layer which enables a simple transmission of TCP/IP protocols over the Packet Data Channel. It can be used simultaneously with the MHP and is distinguished via the TelID (see fig. 5.19). MAMAC maps Ethernet versions 1 and 2, as well as SNAP and LLC directly onto the Packet Data Channel. It is supported by MOST25 and MOST50. Further details can be found in [MAMAC].

6 Physical Layer

The MOST Physical Layer Specification specifies both optical and electrical technologies for the physical layer. The optical transmission layer has now been used several years in the automotive area and is widely spread. It currently uses optical fibers (polymer optical fibers, POF) made of polymethyl methacrylate (PMMA) with a core diameter of 1 mm as transmission medium, in combination with light-emitting diodes (LEDs) in the red wavelength range as transmitters and silicon photo diodes as receivers. The optical physical layer has been introduced with a data rate of 22.6 Mbit/s [MOST BaPhy], commonly shortened with the term MOST25. The first edition of this book focused on that technology in detail. Higher bandwidth requirements for today's systems made it necessary to further develop the optical physical layer. The MOST Optical Physical Layer Sub-Specification [MOST Phy150] is released as the follow-up specification with an increased bandwidth of 147.5 Mbit/s, which is described in the following sections as the MOST150 technology. In addition to signal requirements, the transmission properties of polymer fibers and the currently used transceiver technologies will be discussed. The chapter concludes with an overview of further developments of electrical and optical physical bus layers.

6.1 Introduction

According to the OSI layer model, the term P*hysical Layer* is characterized as the lowest level of a network. It thus represents the physical connection between two MOST control components (MOST Network Interface Controller). The MOST Physical Layer Basic Specification [MOST BaPhy] defines four specification points SP1, SP2, SP3, and SP4 along this point-to-point connection (see fig. 6.1) and outlines the basic measurement techniques and parameters for different physical layer technologies.

The Specification points SP1 and SP4 describe the electrical signal requirements between a MOST NIC and a converter (e.g., electrical signal levels, electrical ramp response and connection timeout, jitter requirements). The interface properties between a MOST device plug and a wiring harness are defined by the specification points SP2 and SP3. In a system with an optical physical layer, the parameters at SP2 and SP3 are, for example, the wavelength, optical power, optical rise and fall times, and, moreover, the mechanical dimensions of the MOST device plugs.

Before going into detail with respect to the individual interface requirements at the specification points, basic properties of optical fibers and opto electronic transmitters and receivers will be introduced in the following sections. For further reading regarding a detailed description of opto-electronic transmission technologies, refer to [Gus 98], [Vog 02].

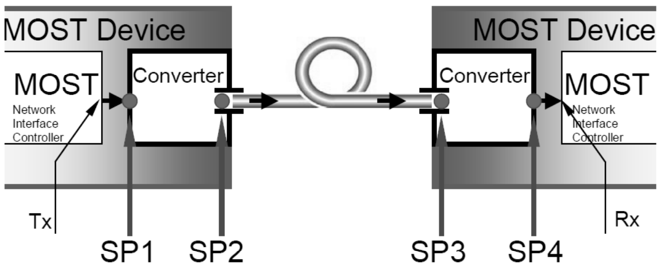


Fig. 6.1: Definition of the specification points SP1, SP2, SP3, and SP4 along a point-to-point connection

6.2 POF Plastic Optical Fibers

Optical fibers are utilized more and more frequently for applications with a high data rate due to increasing EMC requirements (Electromagnetic Compatibility). Transmission lines with optical fibers do not cause any interference radiation and are insensitive to electromagnetic interference irradiation (Electromagnetic Interference, EMI). In contrast to shielded electric data lines, optical fibers are, moreover, lighter and more flexible and can thus be used more easily. The following section describes a number of important basic principles of POF optical fibers for MOST applications.

6.2.1 Basic Principles

Optical fibers consist of a light-conducting core which is surrounded by an optical cladding of a lower refractive index. Figure 6.2 shows a structural diagram of an optical fiber. If a light ray is coupled into the front end of a fiber within the acceptance cone, the light is guided along the core-cladding interface due to total reflectance. The ray travels along a zigzag path and all the light is conducted inside the optical fiber.

The widest angle under which light can be coupled into the front side of an optical fiber is called acceptance angle or numerical aperture. The acceptance angle φ_a and the numerical aperture *NA* are calculated from:

$$NA = \sin\varphi_a = \sqrt{n_{core}^2 - n_{cladding}^2} \qquad \text{(Equation 6.1)}$$

where n_{core} is the refractive index of the optical fiber core and $n_{cladding}$ the refractive index of the optical fiber cladding. The wider the difference of the refractive indices of the core and the cladding material, the wider the acceptance angle or numerical aperture.

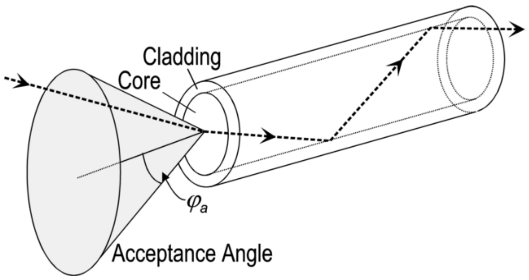


Fig. 6.2: Schematic diagram of light propagation through an optical fiber

Taking the standard polymer fiber for MOST applications with a core refractive index of n_{core} = 1.49 and $n_{cladding}$ = 1.40 as an example, a numerical aperture of 0.5 is resulting, corresponding to an acceptance angle φ_a of 30°. With a high numerical aperture, light can be coupled in more easily. This is particularly interesting if LEDs, which have divergent radiation characteristics, are used as a light source.

A high numerical aperture has, however, the disadvantage of a high mode dispersion. Mode dispersion results from the different propagation delays of light rays, which are coupled in at different angles φ. Light rays with a small angle φ travel faster through the optical fiber than light rays with a wide angle φ. The mode dispersion results in distortions of a rectangular pulse along the optical fiber. Figure 6.3 shows the effects of mode dispersion in a diagram. An ideal rectangular pulse at L_0 expands after a length L_1 by the value $\Delta\tau_1$ and after a length L_2 by the value $\Delta\tau_2$. The longer the optical fiber, the higher the resulting pulse distortion.

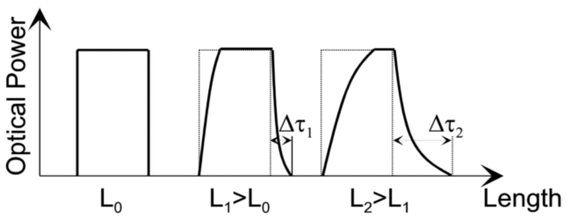


Fig. 6.3: Schematic diagram of pulse widening

For a step-index fiber, the pulse distortion $\Delta\tau$ is calculated according to the formula:

$$\Delta\tau = \frac{n_{core}(n_{core} - n_{cladding})}{n_{cladding} \cdot c_0} \cdot L \qquad \text{(Equation 6.2)}$$

where c_0 is the speed of light and L is the length of the optical fiber. The pulse distortion limits the maximum bit rate of the wave guide. The maximum bit rate can be approximated by means of the equation:

$$B = \frac{1}{\Delta\tau} \qquad \text{(Equation 6.3)}$$

Taking a standard polymer fiber for MOST applications with the core refractive index of $n_{core} = 1.49$ and $n_{cladding} = 1.40$ as an example, one gets a pulse distortion of about 320 ps for a length of one meter. Connections in a vehicle usually have a maximum length of 20 m, which, according to theory, results in a maximum permissible bit rate of 150 Mbit/s. In practice, the bit rates can be higher when transmitting signals via polymer fibers. This is due to the fact that not all possible light rays with different launching angles need to be stimulated when the radiation source emits directed radiation. In the most favorable case, only rays with a small angle φ are present, which results in a lower mode dispersion. An overall conclusion regarding the maximum transmittable bit rate is thus not easy. In the case of specified stimulation characteristics, bit rates of more than 500 Mbit/s can be reached for fiber lengths of about 20 m [Dau 01].

6.2.2 Polymer Fibers – Properties and Advantages

The properties of different core and cladding materials of polymer fibers for different applications have been examined since the late 60s [Wei 99]. Fibers with a core of polymethyl methacrylate (PMMA) and a fluorinated acrylate of a lower refractive index extruded thereon, are used both as light guides for lighting and for short-range transmission systems with a length of less than 100 m. In short transmission ranges polymer fibers with a 1 mm diameter have advantages over glass fibers:

- Flexibility:
  Plastic fibers are much more flexible compared to glass fibers of the same diameter. Fibers with a large core diameter of, e.g., 1mm have good bending properties due to the high flexibility of the polymer material.
- Large fiber diameter:
  Due to the large fiber diameter and the high numerical aperture of about 0.5, the adjusting tolerances for the transmitter and receiver elements are relatively large. It is thus easy to position plastic optical fibers in front of a transmitter or receiver element, as the tolerance requirements are not very high. Transmitter and receiver modules (transceiver) and plug-in connectors can therefore be

manufactured cost-efficiently using injection molding, which is quite interesting for use in vehicles.

- Easy to process:
  Plastic fibers can be easily processed, as the material is soft. The surface can be finished by cutting, grinding or melting. The processing can be carried out with simple hand tools and is not very time-consuming.
- Resistant to contamination:
  Due to the large fiber diameter, slight contaminations at the end surface of the fiber, such as dust or condensation of humidity, only cause little additional attenuation. Plastic fibers can thus be utilized under rough environmental conditions.
- Low costs:
  All the aforementioned advantages result in lower costs, compared to glass fibers. Particularly in cars, short-range connections of less than 20 m on the basis of polymer fibers have a very good price-performance ratio, since the system costs are basically determined by the price of connectors and transceivers, and by the installation and maintenance efforts.

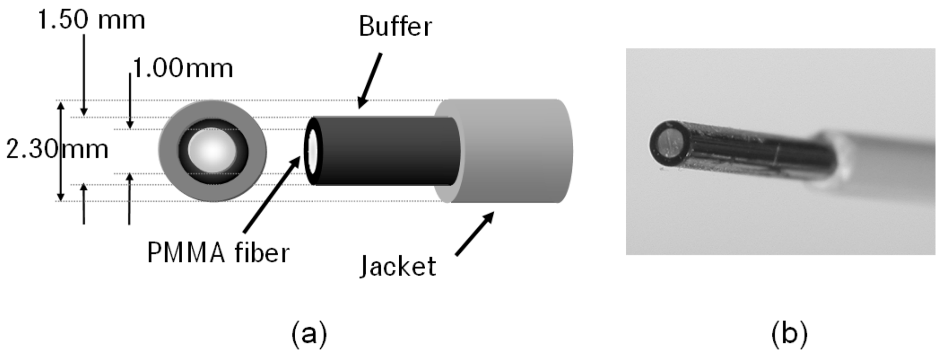


Fig. 6.4: (a) Schematic diagram of the structure of a polymer fiber with buffer and cable coating and (b) the magnified picture of a MOST POF

Figure 6.4 shows the structure of a polymer fiber with protective cladding, according to the requirements for use in vehicles. The PMMA fiber consists of an optical core with a diameter of 980 µm, which is coated with a 10 µm cladding material. The overall diameter of the optical PMMA fiber thus amounts to 1 mm. The optical fiber is surrounded by a black buffer (protective cladding), which is fixed to the optical fiber and cannot be stripped. A high detachment resistance is necessary, since the optical contacts at the end of the fiber are attached to the buffer by means of laser welding or crimping (see section 6.3). A cable coating surrounds the buffer and provides additional protection from mechanical damage and other influences, such as humidity or vehicle operating fluids.

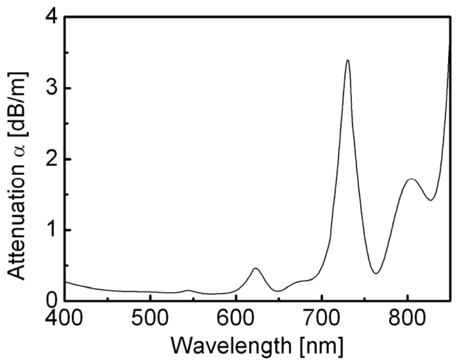


Fig. 6.5: Typical attenuation spectrum of a PMMA fiber

Optical attenuation is an important parameter of an optical fiber. The attenuation of an optical fiber is usually specified by the logarithmic attenuation coefficient α as dB/m. Figure 6.5 shows the attenuation of a PMMA fiber depending on the wave length. The attenuation spectrum of a standard step-index (SI) PMMA fiber has three minima at 520 nm, 570 nm and 650nm in the visible wavelength range. MOST applications use the window in the red wavelength range at 650 nm, as low-priced transmitter and receiver elements are available for this wavelength. The attenuation minimum at 650 nm is 0.14 dB/m.

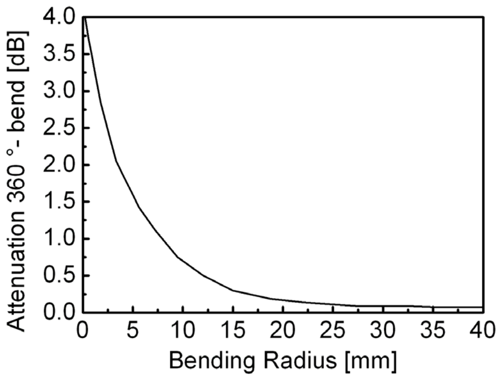


Fig. 6.6: Typical graph of the attenuation of a full launched PMMA fiber depending on the bending radius

Standard PMMA harness fibers are qualified to a temperature range up to 85 °C. For that reason, the use of standard PMMA fibers in vehicles usually is limited to the interior. Since the introduction of optical fibers for MOST, the allowed temperature range could be increased up to 105 °C. However, it may be necessary to use heat protection in the installation spaces, where higher temperatures occur. Polycarbonate fibers were developed for even higher temperature requirements and can endure up to 135 °C. The attenuation of these fibers, however, is higher; it is 0.4 dB/m at the minimum [Wei 99].

Next to deterioration possibly due to aging, the attenuation of an optical fiber during operation is mainly influenced by additional losses through bending. Figure 6.6 shows the attenuation of a PMMA fiber depending on the bending radius. At a bending radius of less than 10 mm, the attenuation rises by more than 0.5 dB compared to the basic attenuation. At a bending radius of more than 25 mm, there is virtually no rise in attenuation due to bending.

Note: In order to prevent additional losses in a vehicle connection, the bending radius shall not go below 25 mm for general applications. This also ensures that the fiber is not damaged by too sharp bending when the connection is manufactured and installed into a vehicle.

6.3 Contacting of PMMA Fibers

Figure 6.7 gives examples of different contacting technologies for PMMA fibers. For the crimp method, a metallic ferrule (usually made of brass) is mechanically pressed onto the buffer. For the ultrasonic welding procedure, a plastic ferrule is melted onto the buffer. For serial production it is very attractive to attach the ferrule by way of laser welding. A laser beam melts the buffer material with the ferrule material. This makes it necessary to use a ferrule material that is transparent to the laser beam so that the welding point can be set between buffer and ferrule. In order to reach a high absorbance at the buffer material, carbon particles are blended in when the buffer is extruded. It is particularly important for high processing reliability of laser welding that the laser welding beam be highly stable and that the absorbance of the buffer material and the transparency of the ferrule are highly homogeneous.

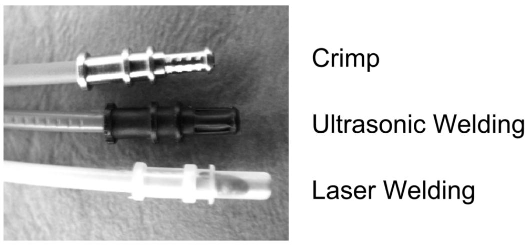


Fig. 6.7: Contacting technologies for polymer fibers (source: Tyco Electronics)

Hand tools are available on the market for assembling optical fibers by means of crimping. Figure 6.8, left side, shows tools for stripping and cutting an optical fiber. A metallic ferrule is subsequently attached to the fiber end by a crimping tool, as shown in figure 6.8 at the right side.

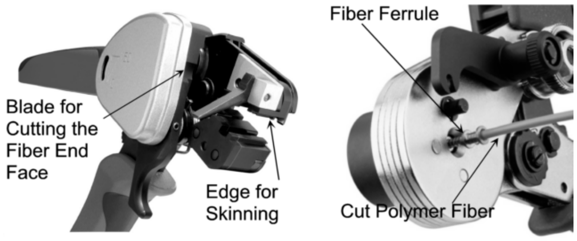


Fig. 6.8: Hand tools for assembling optical fibers (source: Rennsteig Werkzeuge)

A number of manufacturers offer fully automatic machines for manufacturing laser welded ferrules for serial production. Figure 6.9 shows an example of a fully automatic machine with several processing steps for cutting the POF fibers according to the customer's requirements, stripping the ends of the fibers and attaching the plastic ferrules by means of laser welding after finishing the end-face of the fiber and measuring the attenuation.

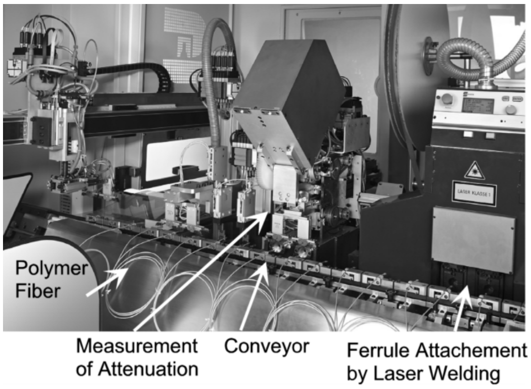

Polymer Fiber
Measurement of Attenuation
Conveyor
Ferrule Attachement by Laser Welding

Fig. 6.9: Fully automatic machine for assembling polymer fibers (source: Schäfer)

The assembled optical fiber cables are delivered to a manufacturer of cable harnesses and are incorporated there into the complete cable harness which is then delivered to a car manufacturer. Specific limiting stresses, such as temperature, humidity and bending radius are to be observed during the entire logistics chain. If required, the optical fibers must be protected by bending radius limiters and dust protection covers (see chapter 16).

The use of inline connectors makes modular construction of the cable harness possible. Figure 6.10 shows an example of an inline connector by means of which two fiber ferrules can be firmly connected by a guide sleeve and a catch mechanism. It is to be considered that including such a kind of disconnection point may cause additional losses of up to 2 dB due to an air gap, an axial displacement and Fresnel reflections at the end faces of the fibers.

Note: If inline connectors are used, the additional attenuation of 2 dB has to be accounted for when designing the system (see section 6.5.3).´

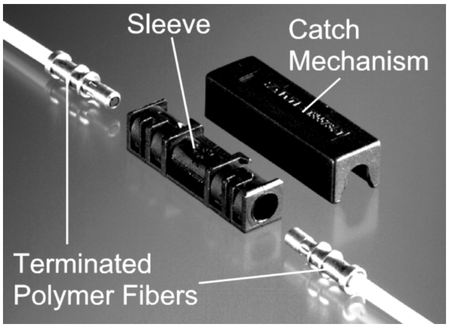


Fig. 6.10: Inline connector for optical fiber disconnection points (source: Tyco Electronics)

6.4 Opto-Electronic Transceivers

Ever since MOST was established, LEDs (Light-Emitting Diodes) have been used as transmitter units in the red wavelength range for converting electrical signals into optical signals. PIN photo diodes are used as receivers. A component which incorporates both transmitter and receiver is called a transceiver (a fusion of the words transmitter and receiver). The following sections will go into detail with regard to basic properties of LEDs and PIN photo diodes. In addition, typical packaging designs of MOST transceivers will be introduced.

6.4.1 LED Transmitter Devices

Light-emitting diodes having an emission wavelength of 650 nm are attractive transmitter units for cost-efficient PMMA networks of less than 100 MHz bandwidth. At 650 nm, LEDs are usually made from the compound semiconductor AlInGaP.

Figure 6.11 shows the schematic structure of a typical LED. On applying a voltage in the forward direction, photons are generated at the p-n junction by recombining carriers. The direction of the generated photons is statistically distributed, which results in the fact that not all photons are coupled out at the surface of the semiconductor crystal. The efficiency of a standard LED is thus only 2-4 %. It is possible to considerably increase the external efficiency of LEDs by means of suitable structuring measures and suitable packaging. LEDs are often designed into reflectors, in order to utilize the rays emitting at the side. Mirror structures at the backside of LEDs are used to reflect the rays hitting the base of the substrate back to the surface. The LED surface may be roughened or covered by a polymeric hood, which results in higher out-coupling. If the necessary measures are taken, the external efficiency reached today can amount to more than 20 %.

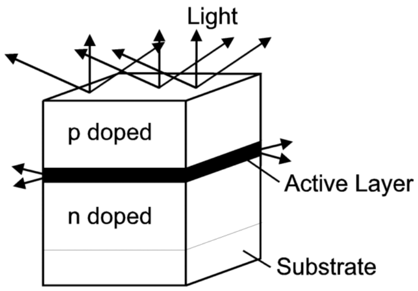


Fig. 6.11: Schematic diagram of an LED

At a forward current of between 10 and 40 mA the optical power of an LED for applications in data communication is typically between 0.1 and 1 mW. It is often characterized by the logarithmic unit dBm (related to 1 mW optical power) and is calculated by way of the following equation:

$$P_{dBm} = 10 \cdot \log\left(\frac{P_{mW}}{1\text{mW}}\right)\text{dBm} \qquad \text{(Equation 6.4)}$$

The optical power of 1 mW thus corresponds to 0 dBm; the optical power of 0.5 mW amounts to -3 dBm, which is 3 dB less.

The modulation bandwidth of a commercially available LED amounts to a maximum of about 100 MHz. The bandwidth can be increased by changing the doping concentration in the active layer. However, the output power is further decreased due to the inserted impurities. Another possibility of increasing the bandwidth is *peaking*, as it is called, where the driving current at the start of a data pulse is raised, thus achieving a shorter rise time. However, complex control circuits are required for that purpose in order to be able to control the overshooting behavior. It seems that bandwidth and efficiency cannot be optimized at the same time.

In the early 90s resonant cavity LEDs (RCLEDs) were introduced for the first time. In RCLEDs the light-emitting active layer is embedded between a short resonator, by which an increase of efficiency and modulation bandwidth can be achieved. Components with both higher bandwidths and a higher output performance can thus be realized. Due to their specific structure, the performance of RCLEDs is, however, more sensitive to temperature than the performance of LEDs, which must be compensated for by a suitable driver. Recently manufacturers have started to offer suitable driver-RCLED combinations [DS FCM110] which are designed for MOST applications.

Due to their high efficiency and a modulation bandwidth of far more than 500 MHz, red laser diodes would be well suited for PMMA networks. At present, however, the admissible surrounding temperature of available red laser diodes is restricted to ca. 70 °C, which would require a complex component cooling, if these diodes are used.

6.4.2 PIN Photodiodes

Silicon-PIN photodiodes are used as cost-efficient receiver components for converting optical signals into electrical signals in a wavelength range of 400 nm to 1100 nm. Figure 6.12 shows the schematic structure of a PIN photodiode on the left side and the responsivity R_{Ph} along the wavelength on the right side. The responsivity increases virtually linearly from 0.32 A/W at 500 nm to 0.7 A/W at 900 nm. At 650 nm the responsivity is ca. 0.47 A/W. Large-area diodes are necessary for efficiently converting light of PMMA fibers with a core diameter of 1 mm. A large receiver area, however, also means a high capacity, which in turn reduces the bandwidth. With bandwidths of up to 100 MHz and photo diodes of 1 mm diameter a receiver sensitivity of less than -30 dBm can be achieved.

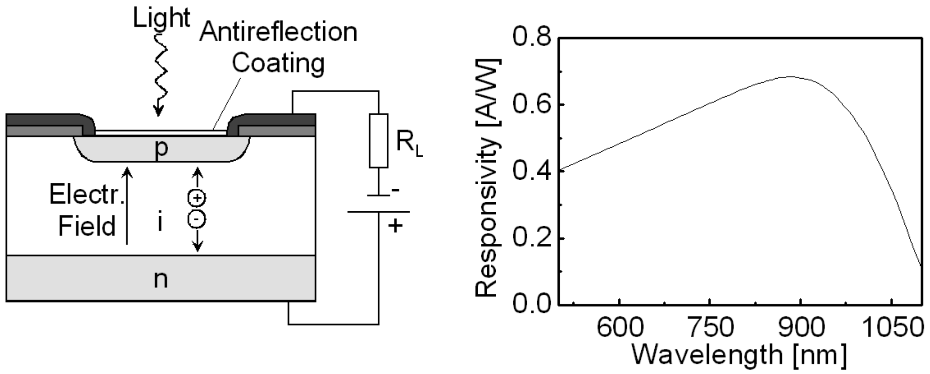


Fig. 6.12: Schematic diagram of a PIN photodiode and typical gradient of the responsivity of a silicon photodiode

6.4.3 Transceiver Housing and Connectors

A couple of manufacturers have developed transceivers (Fiber Optic Transceiver, FOT) with integrated opto-electronic components for MOST networks. Figure 6.13 shows an example of an LED transmitter and a PIN photodiode receiver for MOST. The LED and the driver component, or the photodiode and the receiver amplifier as the case may be, are attached onto a leadframe and encapsulated in a transparent plastic material. Seven electrical connecting pins lead out of the plastic housing to provide the contacts for current supply, differential data signals and other status or control lines. The MOST Physical Layer Sub-Specification [MOST Phy150] describes this kind of package THM (Trough Hole Mount), because the opto-electronic components are mounted through PCB holes. The handling of the THM package is well known from the MOST25 technique and appropriate standard processes have been established at the component manufacturers.

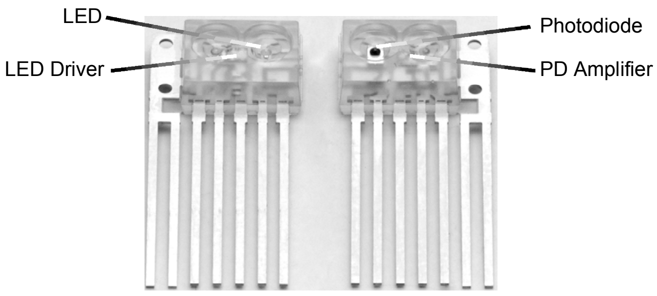


Fig. 6.13: Transparently cast transmitter and receiver elements (source: Hamamatsu Photonics)

The assignment of the individual electrical connecting pins is defined by the MOST Specification. Figure 6.14 shows the used pin assignment. Both transmitter and receiver components each have supply connections (V_{cc}, with 3.3 V) and a ground connection (GND). Besides the differential data input (Tx+ and Tx-) and data output (Rx+ and Rx-), the Reset (\RST) pin at the transmitter allows to switch on and off the optical output at a defined time during startup and shutdown to avoid optical glitches. The status pin at the receiver is used to set the MOST Network Interface Controller into a ready-to-receive state as soon as the receiver receives data signals. Further information concerning the electric circuit of the components can be gathered from the data sheet of the respective manufacturer.

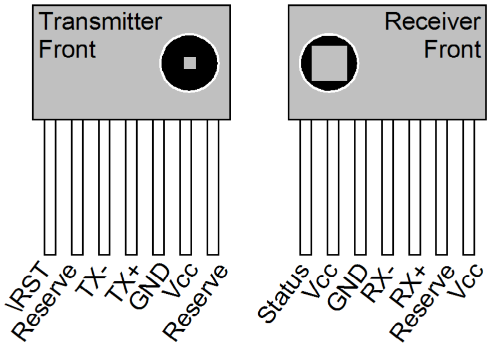
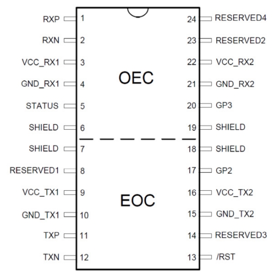


Fig. 6.14: Typical pin assignment of a THM packaged FOT

Since the introduction of the MOST150 technology, there is also an SMD packaged transceiver available to be compatible with state-of-the-art assembly and soldering processes. The SMD package shows better EMI performance, compared to the THM package, and is basically qualified for even higher data rates up to the gigabit region. Furthermore the use of the SMD technology results in more flexibility in the supply chain, because it is no longer necessary to purchase the FOT together with the connector and pigtail. The opto-electronic component can be ordered directly from the FOT vendor with huge cost benefits.

Figure 6.15 shows the pin assignment and a sample of a SMD packaged transceiver. The transceiver package is based on the SOIC (Small Outline Integrated Circuit) standard. To contact the fiber in the vertical direction a special spring mechanism is specified. The spring forces are defined to have both a soft plug-in for the assembly and a sufficient holding force to confirm a stable connection of the fiber.



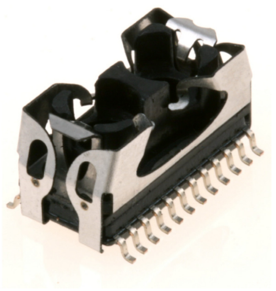


Fig. 6.15: SMD transceiver pinout and sample device (source: Melexis).

For both the THM packaged transceiver and the SMD packaged transceiver, there are device connectors available to offer the end user a standardized connector interface for MOST. Figure 6.16 shows schematic drawings of device connectors, characterized as 2+0 and 4+40.

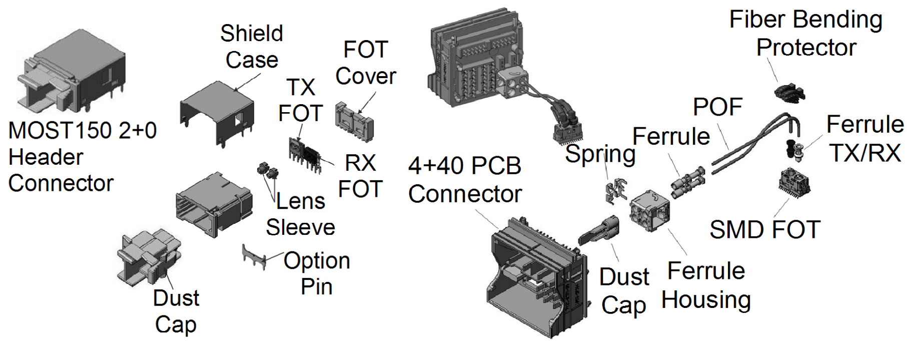


Fig. 6.16: Schematic and exploded drawing of a MOST150 2+0 header connector (left) and a MOST150 4+40 (right) header connector (source: Yazaki Corporation)

The designation indicates the number of optical and electrical contacts. The 2+0 connector, shown in the left hand side has two optical contacts (one for the transmitter and one for the receiver), and no electrical contacts. The 2+0 header connector features a compact design to connect the THM packaged transceivers to the wiring harness, as well as low costs. The 4+40 connector, shown in Figure 6.16 on the right hand side has four optical contacts, and 40 electrical contacts. Between the SMD packaged transceiver and the device connector a flexible pigtail is used. In the case of the flexible pigtail, the FOTs are outside the actual connector housing of the control device. This has the advantage that the active components can be randomly placed on the circuit board of the control device. It is thus easy to place the FOTs near the MOST Network Interface Controller and at the same time avoid electromagnetic interference problems. The pigtail consists of a standard high temperature polymer fiber with attached transceiver ferrules. A maximum insertion loss of 1.5 dB is allowed, including fiber attenuation, fiber mismatch and surface roughness. A bending protector could be applied to the pigtail fiber to prevent a too low bending radius

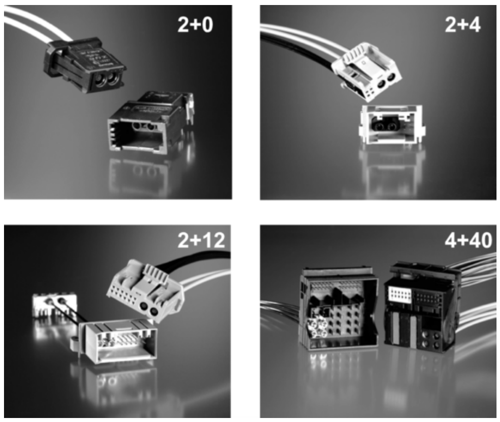


Fig. 6.17: Connector variants for MOST interfaces (source: Tyco Electronics)

Figure 6.17 shows an overview of typical standard interfaces for MOST. The assignment of the optical contacts is defined by the MOST standard, whereas the electrical contacts can be defined by the end user as desired. The MOST standard only defines the connector housing at the control device. The connector at the wire harness is not part of the standard. Due to the nature of the wire harness connector, however, a maximum permissible optical insertion attenuation of 2.5 dB is to be observed. A detailed definition of the device connector geometry can be gathered from the Interface-Drawings of the MOST Physical Layer Sub-Specification [MOST Phy150]. Further requirements, such as the plug-in or unplug force are typical of a specific end user and thus not part of the MOST standard. Apart from the connector groups showed in figure 6.16, the MOST standard lists a 2+20 alternative.

The individual parts of a control device connector and a wire harness connector are illustrated in example 6.18. The exploded drawing shows a 2+0 control device connector and a 2+0 wire harness connector, where the fiber ferrules are connected in an internal housing. As required, a bending protector for the polymer fiber can be attached to the wire harness connector so that the minimum admissible bending radius of 25 mm is observed. The individual parts of the interfaces are sold separately by most connector manufacturers. Wire harness connectors with optional line lengths can thus easily be produced by the users themselves. If it needs repair, the wire harness connector can be disassembled into its individual parts with little effort.

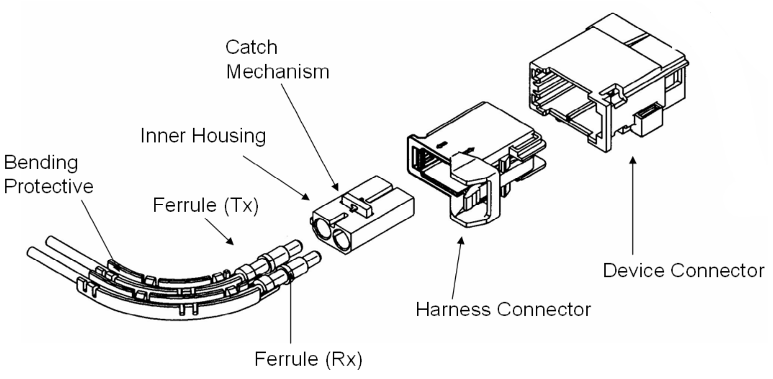


Fig. 6.18: Exploded drawing of a 2+0 connector housing with wire harness connector (source: Yazaki Corporation)

6.5 System Considerations for the MOST Physical Layer

This section describes the electrical and optical parameters and requirements of the physical bus layer. Before explaining the exact specification parameters, a number of basic parameter definitions are stated. The section will be concluded by considering power budget and timing.

6.5.1 Basic Principles of Electrical and Optical Parameters

Data Rate and Coding

The data rate of MOST150 systems results from the product of the frame size (BPF, bits per frame) of 3072 bits and the frame rate (*Fs*, frame synchronization). At the typically used sampling rate 48 kHz, the required bit rate is 147.56 Mbit/s. The pulse length UI (unit interval), which is used as the basic pulse length unit, is defined as the half bit transfer time and can be calculated with the equation 6.5:

$$UI = \frac{1}{2 \cdot Fs \cdot BPF} \quad . \qquad \text{(Equation 6.5)}$$

At a frame rate *Fs* = 48 kHz and the number of bits per frame (*BPF* = 3072), one unit interval is 3.391 ns. The MOST150 encoding is according to a DC adaptive coding (DCA). The DCA coding compensates DC offsets by limiting the accumulated DC offset contained in the data. Five different symbol lengths are defined, while the shortest pulse is 2 UI and the longest 6 UI. Depending on the data content there are 2, 3, 4, 5, and 6 UI but never 1 UI pulses (see fig. 6.19). By avoiding the

1 UI pulse length, the needed transmission bandwidth is reduced. The 6 UI pulses only appear as preamble at the beginning of a new data frame and serve for synchronization.

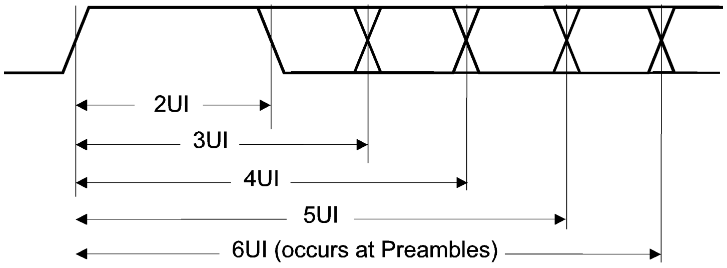


Fig. 6.19: Schematic view of the possible pulse length for MOST150

Figure 6.20 gives a schematic view of the influences of various effects which result in a modification of the ideal rectangular pulse. The pulse width can differ slightly from the 2, 3, 4, 5, and 6 UI widths. Due to variations of temperature, noise, nonlinearities of drivers, etc., duty cycle variations (pulse width variations) jitter (phase distortions) is generated. The bandwidth of all components in the transmission sector limits the rise-time and fall-time of a pulse, and the control in driver and amplifier components causes overshoot and undershoot in the signal.

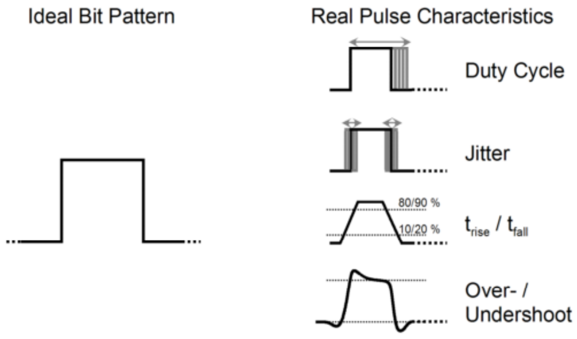


Fig. 6.20: Schematic diagram of ideal and real pulse shapes

The MOST Specification defines limit values for the maximum deviations from the ideal signal at the interfaces SP1, SP2, SP3 and SP4 in order to reach an error rate of less than 10 ^{9}. Since the time-critical signal parameters are influenced by many factors, these effects will be described in detail in the following.

Timing Distortions

The MOST Specification combines the time-critical parameters under the generic term *timing distortions* according to figure 6.21. The timing distortion *alignment jitter Aj* and *duty cycle* are effects which are characteristic for a single link. They are relevant if the signal integrity at the specification points SP1, SP2, and SP4 is measured with an eye diagram. The alignment jitter is the jitter relative to the recovered clock. In MOST150 systems, the alignment jitter is observed for a bandwidth larger than 125 kHz, because the Phase-Locked Loop (PLL), used in the MOST Network Interface Controller, is able to track and eliminate phase variations up to this frequency. To measure alignment jitter, the MOST specification defines a "Golden PLL" model, given in the form of a low-pass transfer function with a cut-off frequency of 125 kHz. The timing distortions *system clock deviation*, *wander* and *transferred jitter Tj* describe modifications in the timing which can propagate over one or several active nodes. The accumulation of these timing distortions may not hit the system margin for the Master Delay Tolerance (see section 6.5.4) and therefore limits the maximum number of nodes in the network. Due to the low-pass characteristic of a NIC, the jitter transfer in each node is bandwidth limited. Because of that, the NIC has an effect of a "Jitter Filter" and is able to eliminate jitter at a frequency up to 200 kHz.

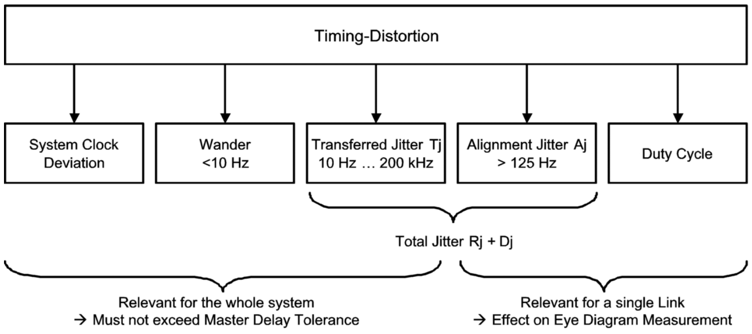

Timing-Distortion
System Clock Deviation
Wander <10 Hz
Transferred Jitter Tj 10 Hz … 200 kHz
Alignment Jitter Aj > 125 Hz
Duty Cycle
Total Jitter Rj + Dj
Relevant for the whole system → Must not exceed Master Delay Tolerance
Relevant for a single Link → Effect on Eye Diagram Measurement

Fig. 6.21: Overview of the timing-distortions

The MOST25 Physical Layer Specification defined maximum values for each timing distortion by a separate parameter. However, this specification did not allow sharing system reserve between different types of timing distortions. Furthermore, low bandwidth distortions, which basically can be eliminated by PLL tracking, made measurement results worse. For MOST150 systems, these issues have been solved by using an eye diagram measurement and PLL triggering.

Eye Diagram and PLL Triggering

An eye diagram shows an overlay of all pulses a digital signal takes on during a bit period (UI). The bit period is defined by the data clock, which is necessary to measure the eye pattern. Usually an eye diagram is acquired by using a reference clock as a trigger. If the reference clock is not available as a discrete signal, a virtual reference clock can be recovered by using the PLL trigger method of a serial data analyzer. With each virtual trigger, the signal is sampled and stored in a persistence map.

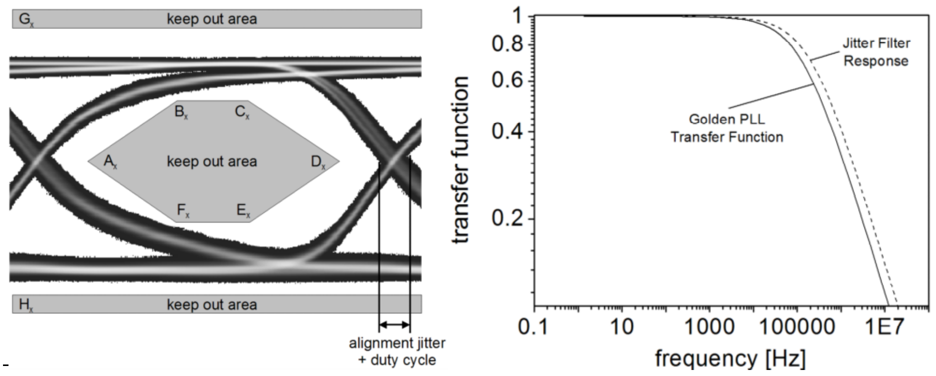
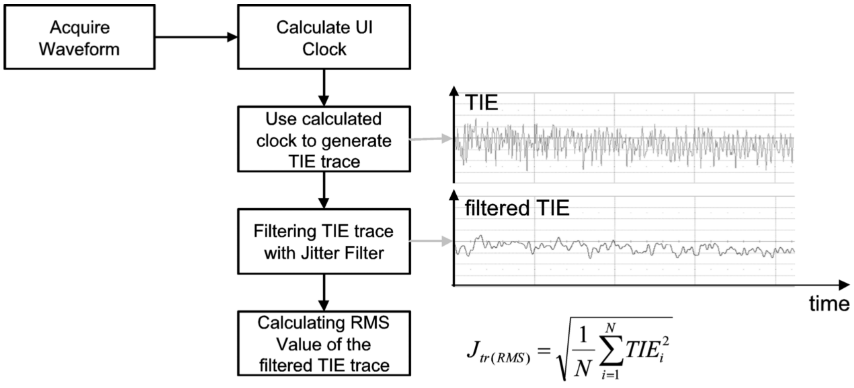


Fig. 6.22: Example of an eye diagram measurement, indicating the eye mask (left) and definition of the transfer function for the Golden PLL and the Jitter Filter (right).

Figure 6.22 shows an example of an eye diagram on the left hand side, measured with a PLL model according to the MOST150 “Golden PLL”, indicated on the right hand side of Figure 6.22. The advantage of eye diagram measurement is that the timing tolerances for all kinds of timing distortions are shared. That means good duty cycle conditions allow more alignment jitter or low jitter allows more duty cycle. The signal quality can be evaluated by comparing the acquired eye diagram with an eye mask. The masks are keep out areas that shall not be touched by the signal trace. For the eye diagram measurement, different masks for SP1, SP2, and SP4 are specified. Figure 6.22 indicates the specification parameters A_x, B_x, … F_x for the link quality measurement, with the index x = 1, 2, 4 according to the specification point (SP) under investigation. To prevent exceeding the maximum voltage levels at SP1 and SP4, further mask definitions with the parameters G_x and H_x have to be considered. Details can be found in the Optical Physical Layer Sub-Specification.

For a MOST system with several nodes, the transferred jitter is significant. Jitter can be passed on from slave to slave and possibly become so high that the NetworkMaster can no longer synchronize the MOST signal correctly due to the degradation. It is therefore important for the entire system that the sum of all jitter and delay effects does not exceed the Master Delay Tolerance, which is discussed in detail in section 6.5.4. Figure 6.23 shows the procedure to measure transferred jitter. After acquiring the waveform, the time-interval error (TIE) is calculated by the phase deviations between the waveform and the best matching fix clock signal. The TIE graph then is filtered with the low-pass Jitter Filter characteristic, given in Figure 6.22 on the right hand. The parameter transferred jitter J_{tr} finally is calculated as the root-mean-square sum of the filtered TIE graph. The edge frequency for the Jitter filter is set to the highest expected PLL performance, to include the whole frequency range that might be transferred trough a MOST node. For the acquisition of the waveform, a scope with a deep memory of at least 10 Msamples is needed to measure also the low bandwidth jitter portion. For example, with a memory depth of 10 Msamples and a sampling rate of 10 Gsamples/s the maximum record length is 1 ms. This record length allows measurements down to the frequency of 1 kHz. The transferred jitter is specified as a RMS value for SP1, SP2 and SP4.


Acquire Waveform
Calculate UI Clock
Use calculated clock to generate TIE trace
Filtering TIE trace with Jitter Filter
Calculating RMS Value of the filtered TIE trace
TIE
filtered TIE
time

$$J_{tr(RMS)} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} TIE_i^2}$$

Fig. 6.23: Procedure to measure the transferred jitter J_{tr}

Optical Power

An important parameter for characterizing MOST components is the optical power of a MOST transmitter (SP2) or the power to be coupled into a receiver (SP3). Figure 6.24 shows an optical pulse diagram and the corresponding definition of the optical power levels. The average optical power is thus defined by the parameter P_{AV} (average power). The parameters *optical high level b_1* and *optical low level b_0* describe the levels of the *1* and *0 states*. The level b_0 is not equivalent to "light off". Due to the extinction ratio $r_e = 10 \log (b_1/b_0)$ dB, the specification indicates, however, the minimum ratio between the two levels. For MOST150 it is specified to measure the b_0 and the b_1 level at the shoulder of 5 or 6 UI Pulses within a measurement region between 2.5 and 4 UI, as shown in Fig 6.24.

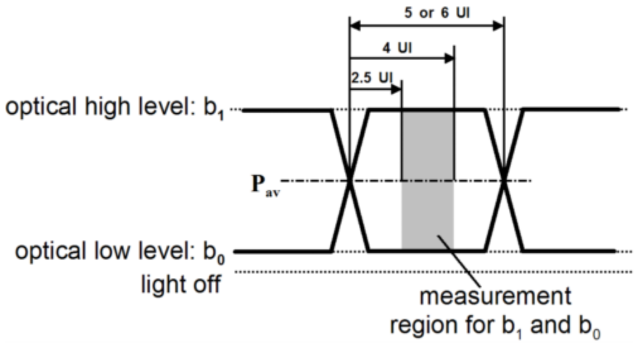

5 or 6 UI
4 UI
2.5 UI
optical high level: b_1
P_{av}
optical low level: b_0
light off
measurement region for b_1 and b_0

Fig. 6.24: Definition of the optical levels b_1, b_0 and P_{AV}

Measurement Tools

For measuring the electrical signal parameters at SP1 and SP4, an active probe with a high bandwidth (>1,5 GHz), a minimum input capacity and a maximum input resistance is required, in order to prevent any distortion of the original signal. The optical specification points SP2 and SP3 shall be evaluated with a high speed OEC with a bandwidth >750 MHz and a photo diode with a large active area is required, in order to capture enough light of the 1 mm PMMA fiber. In addition, the OEC shall have a signal-to-noise ratio (SNR) better than 8:1. For measurements in the field, the average optical power P_{AV} is usually measured by a simple power measuring device. Various manufacturers offer calibrated portable measuring devices for that purpose. It has to be observed that only the optical power at SP2 and SP3 is measured which is emitted at a diameter of 1 mm and within a numerical aperture of 0.5. Further information on this matter can be gathered from the MOST150 oPhy Compliance Measurement Guideline [ComPhyL 150].

6.5.2 Optical and Electrical Requirements

The MOST Physical Layer Sub-Specification [MOST Phy150] defines the required parameters for all specification interfaces SP1, SP2, SP3 and SP4, in order to ensure a reliable transmission of data. The automotive area demands that all parameters are observed, even under the most unfavorable environmental conditions. The environmental conditions to be considered are always specific to a particular end user. Some car makers got together and developed a recommendation titled *Automotive Application Recommendation for optical MOST Components* [MOST AARTHM], [MOST AARSMD] for harmonizing the environmental requirements. Typical requirements are a temperature range of -40 °C to 95 °C, a humidity resistance of up to 85 % at 95 °C and mechanical robustness during an operation period of ca. 11,250 hours and a life cycle of 15 years. The required tests (temperature cycles, humidity cycles, mechanical impacts, EMC/EMI) for the qualification of a MOST product are exactly defined in the application recommendations.

Electrical Specification at SP1

The signal levels at SP1 are defined by the LVDS standard. The transmission is based on differential signaling, to avoid electromagnetic interference. The signal has to fulfill all required parameters like alignment jitter, transfer jitter and the amplitude values. In addition, a new circuitry is defined for MOST150 to achieve a well-regulated startup and shutdown procedure.

Optical Specification at SP2

The peak wavelength of 650 nm corresponds to the attenuation minimum of PMMA fibers in the red wavelength range. Due to the fact that the attenuation of PMMA fibers rises below and above 650 nm, the admissible wavelength range is restricted to between 630 nm and 685 nm. For the same reason, the width of the spectrum is restricted to 30 nm (FWHM).

The optical spectrum for a Gaussian distribution is defined via the peak wavelength and the width FWHM. This definition is not suited for asymmetrical spectra, as no clear peak wavelength can be defined. Figure 6.25 compares the symmetrical wavelength spectrum of an LED to an asymmetrical spectrum. Asymmetrical spectra are evaluated by an alternative definition via the central wavelength λ_{c2} and the spectral width (RMS) $\sigma_{\lambda 2}$. The central wavelength and the spectral width can be calculated from a measured spectrum. The center wavelength is calculated from:

$$\lambda_{c2} = \frac{\sum_i P_i \lambda_i}{\sum_i P_i} \quad \text{(i=500 nm…800 nm, } \Delta i \leq 1 \text{ nm)} \qquad \text{(Equation 6.6)}$$

where P_i is the optical power at a specific wavelength λ_i. The spectral width is calculated from the following equation:

$$\sigma_{\lambda 2} = \sqrt{\frac{\sum_i P_i (\lambda_i - \lambda_{c2})^2}{\sum_i P_i}} \quad \text{(i=500 nm…800 nm, } \Delta i \leq 1 \text{ nm)} \quad \text{(Equation 6.7)}$$

For a sufficiently exact measurement, the spectrometer used must have a signal-to-noise ratio of at least 200:1 and a resolution of less than 1 nm. In addition, the spectrum in the entire wavelength range from 500 nm to 800 nm is to be evaluated, as the determination of the center wavelength and the spectral width becomes otherwise too inexact.

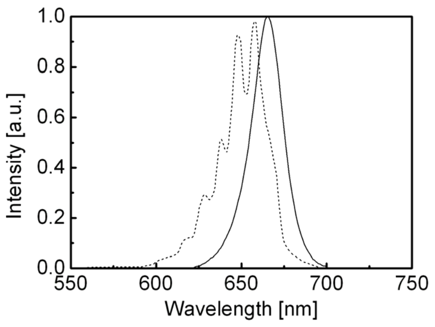


Fig. 6.25: Gaussian spectrum (continuous line) and asymmetrical spectrum (broken line) of light-emitting diodes

The optical power at SP2 is specified between -8.5 dBm and -1.5 dBm. The lower limit may not be less, so to ensure data recovering at the receiver. The upper limit is defined in order to avoid overload or blinding the receiver. The parameter P_{OFF2} of -50 dBm defines the maximum admissible power level for the *light off* state. It is necessary to observe this level in order to be able to reliably switch off a subsequent receiver. Even if no more optical alternating signals are transmitted, some receiver components react to a minimal amount of d.c. light and prevent the switch-off procedure (sleep-mode) of the MOST bus. The extinction ratio must be at least 10 dB so that the responsivity of the receiver is not impaired.

The other timing parameters (transition, alignment, and transfer jitter) are slightly relaxed compared to the specification at SP1, giving margin for slight distortions during the electro-optical transformation. Additional definitions at SP2 are given by the parameters *positive overshoot*, *negative overshoot* and *high-level signal ripple*. These parameters limit the amount of overshoot in the optical signal. For a graphical summary of these parameters, refer to the figure in [MOST Phy150].

Optical Specification at SP3

The definition of the optical spectrum remains unchanged compared to SP2. The dynamic range of the link is indicated by the parameter P_{opt3} ranging from -22 dBm to -2 dBm.

For the optical data bus system with a data rate of around 150 Mbit/s, the bandwidth limitation of the polymer optical fiber, discussed in section 6.2.1 *Basic Principles* becomes considerable. The bandwidth limitation results in variations of the optical signal in vertical (amplitude) and horizontal (pulse width) direction. The MOST Physical Layer Working Group conduct some research to determine the polymer fiber transfer function. The detailed analysis comes to the result that with a maximum link length of 15 m and a full launched fiber the -3 dB cut-off frequency is about 90 MHz [MOST AppTF]. The highest frequency of a MOST150 data pattern is 75 MHz. Therefore the system is close to the bandwidth limit but can fulfill the required signal integrity if the link length does not exceed 15 to 20 m. The worst case input signal at SP3 can be calculated by the convolution of the worst case signal at SP2 and the POF transfer function with a given link length and launch condition.

Electrical Specification at SP4

Similar to the specification point SP1, the electrical levels correspond to the LVDS standard. The eye diagram of a measured electrical signal must not hit the keep out area of the defined mask, indicating the allowed jitter and the LVDS voltage level. These criteria allow evaluating the link quality, which describes the performance of a single link without any system aspect. For a complete system characterization the SP4 receiver tolerance has to be considered. The applied eye mask for defining the receiver tolerance is much smaller than the link quality eye mask, taking the accumulation of jitter at each node in the network into account.

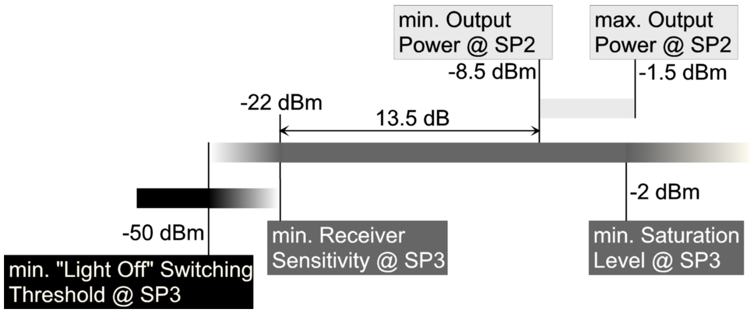

min. Output Power @ SP2
max. Output Power @ SP2
-8.5 dBm
-1.5 dBm
-22 dBm
13.5 dB
-2 dBm
-50 dBm
min. Receiver Sensitivity @ SP3
min. Saturation Level @ SP3
min. "Light Off" Switching Threshold @ SP3

Fig. 6.26: Power budget of an optical MOST system between SP2 and SP3

6.5.3 Power Budget

The minimum permitted transmit power at SP2 and the receiver responsivity at SP3 determine the available optical power budget. Figure 6.26 shows the power budget between the two interfaces SP2 and SP3. According to the illustration, a power margin of 13.5 dB is calculated. Due to the fact that the insertion attenuation of MOST connectors at the interfaces SP2 and SP3 of the control devices is 2.5 dB each, the margin available for the wire harness is reduced to 8.5 dB.

The attenuation of the entire optical connection between two control devices must not exceed the wire harness reserve of 8.5 dB. Attenuations on the transmission link result from fiber attenuation, inline connectors and possibly from bending and ageing losses.

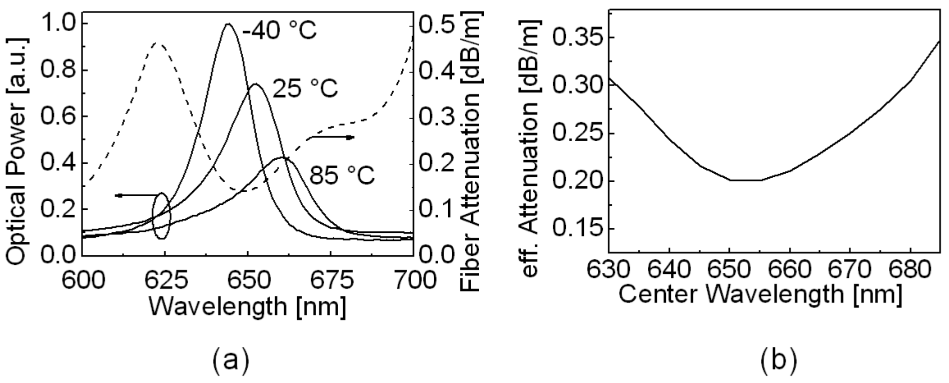


Fig. 6.27: (a): Spectral attenuation of a PMMA POF and spectral output power of a red LED at varying temperatures
(b): Effective fiber attenuation depending on the center wavelength. The spectral width of the transmit source is l = 30 nm.

As already mentioned in section 6.2.2, the fiber attenuation of a PMMA fiber is 0.14 dB/m at a wavelength of 650 nm. Both, the temperature dependent output power and emission spectrum of LEDs, however, strongly influence the power budget of the transmission system. Figure 6.27 on the left shows a detail of the attenuation spectrum of a PMMA fiber between 600 nm and 700 nm and the emission spectrum of a red LED at varying temperatures. The illustration clearly shows that a higher attenuation than at the minimum at 650 nm is applied to a large part of the LED. Apart from that, the optical power decreases with increasing temperature, the center wavelength drifts to higher wavelengths with 0.16 nm/K and the spectral width increases. Due to this wavelength drift a higher effective fiber attenuation α_{eff} must be assumed for the system consideration, according to the equation

$$\alpha_{eff} = \frac{\int \alpha(\lambda) \cdot P(\lambda)}{\int P(\lambda)} d\lambda \qquad \text{(Equation 6.8)}$$

Figure 6.27 shows on the right the effective fiber attenuation of a PMMA fiber calculated from the above equation depending on the center wavelength, calculated for an LED with a spectral width of 30 nm. The effective attenuation amounts up to 0.35 dB/m.

Note: If an additional increase in attenuation due to ageing is taken into account, the resulting effective attenuation of a PMMA fiber is up to 0.4 dB/m. This value must be used for power budget considerations.

At an effective fiber attenuation of 0.4 dB/m and insertion losses of the individual inline fiber connectors of 2 dB at the most, different wire harness concepts and their feasibility, as well as operational reliability including system reserves for assembling, bending, and ageing losses can be calculated.

6.5.4 Timing Requirements

In section 6.5.1 and 6.5.2, the timing requirements for the nodes of a MOST system were described. Compliance with the specified values at SP1, SP2, SP3 and SP4 link quality (LQ) allows the proper operation of an individual point-to-point connection. Table 6.1 shows the specified timing parameters at a bit rate of 147.456 Mbit/s (1UI = 3.39 ns). The highest percentage of phase variation and jitter is caused by the OEC at the receiver interface between SP3 and SP4. The link-level tests use a pattern generator as a signal source. The pattern generator has to provide a test signal according to the MOST150 test pattern, which can be downloaded as a supplement file to the Optical Physical Layer Sub-Specification. To prevent a modification of the test pattern, the Network Interface Controller has to be set in the retimed bypass mode, which deactivates the scrambling mechanism.

Parameters	*SP1*	*SP2*	*SP3*	*SP4 (LQ)*	*SP4 (RT)*
Alignment Jitter	0.15 UI	0.3 UI	-	0.55 UI	0.6 UI
Transfer Jitter	50 psRMS	112 psRMS	-	230 psRMS	-

Table 6.1: Specific timing parameters at SP1, SP2, SP3 and SP4 link quality (LQ) and receiver tolerance (RT) at a bit rate of 147.456 Mbit/s

Additional to the link quality specification parameters, a limit for the SP4 Receiver Tolerance (RT) is specified. The SP4 Receiver Tolerance describes the minimum Alignment Jitter tolerance of a NIC and the maximum tolerable alignment jitter that may occur in any place in the network. The SP4 Receiver Tolerance is measured with a system-level test, which uses live data from a fully formed MOST150 ring. Using the same eye diagram methodologies developed in section 6.5.1, a measurement is taken at SP4 of any node in the ring. By taking the measurement in this way, one can quantify the total jitter accumulation at every position of the ring. This measurement is applicable for every node in the network at SP4.

The timing parameters of real components are, besides the variance due to production, significantly influenced by the received light power at SP3 and the surrounding temperature of the opto-electronic transceiver. Figure 6.28 shows an example of a typical gradient of the timing parameter alignment jitter, depending on the optical power at SP3 at room temperature. In the example given, alignment jitter is quite low in middle optical power range. Getting closer to the specification limit, the jitter increases and the eye closes.

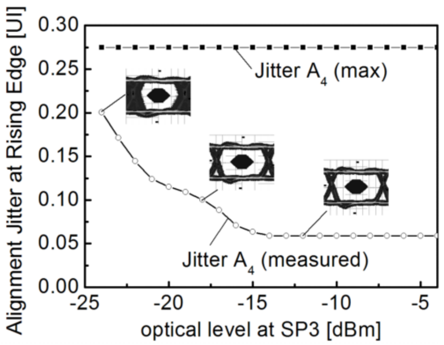


Fig. 6.28: Typical gradient of the alignment jitter at SP4 depending on the light power at SP3

In order to ensure that the entire system consisting of several nodes can operate without errors, the sum of all jitter and delay effects must also be taken into consideration. In terms of the MOST Specification, these effects are described as system delay. The system delay can possibly accumulate to such an extent at the slave nodes in the ring that the TimingMaster can no longer correctly synchronize to the incoming signal. The range in which the TimingMaster can correctly process the incoming signal is called the Master Delay Tolerance. The Master Delay Tolerance is a property of the MOST Network Interface Controller in TimingMaster mode and is listed in its data specification. The maximum value is defined in the Physical Layer Sub-Specification with 0.5*/Fs*. The Master Delay Tolerance of the MOST Network Interface Controller used as TimingMaster can influence the maximum admissible number of nodes in a MOST system. The MOST Physical Layer Sub-Specification [MOST Phy150] defines a maximum node number of N = 20 which is sufficient in the automotive area.

The total Master Delay Tolerance (T_{MDT}) is calculated from the formula:

$$T_{MDT} \geq t_{D\,Rx}(Master) + t_{D\,Tx}(Master) + \sum_{n=1}^{m-1} t_D(n) + \sum_{n=1}^{m} t_W(n)\ldots$$

$$\ldots + t_{D\,Medium} + \alpha \cdot \sqrt{\sum_{n=1}^{m} t_{TJ}{}^2(n)}$$

(Equation 6.9)

The value *m* indicates the number of nodes in the MOST system, $t_D(n)$ the delay per node (caused by the INIC, transmitter, receiver, and medium), $t_W(n)$ the wander (phase drift) per node and link and $t_{TJ}(n)$ the transfer jitter per node. The exact extent of the individual delay values depends on the system and is influenced by the length of the wiring, the characteristics of various transceivers and the working temperature. To evaluate the maximum delay, it is thus necessary to exactly analyze the system under realistic conditions.

Figure 6.29 shows an example of a system with three nodes and the position of the definition points of the individual jitter parts. According to the specification, each point-to-point connection causes a maximum transferred jitter of 230 ps and a maximum alignment jitter of 0.6 UI at SP4 of each node. After the signal of the TimingMaster in the first slave has been transmitted via the PLL to the SP1 interface, the transferred jitter is reduced according to the PLL filter characteristics (<230 ps). The jitter at SP1 of the second slave is made up of the accumulated part of the first and the second slave. Up to the interface SP4 of the TimingMaster, the jitter of the last link finally contributes to the total jitter, system delay respectively.

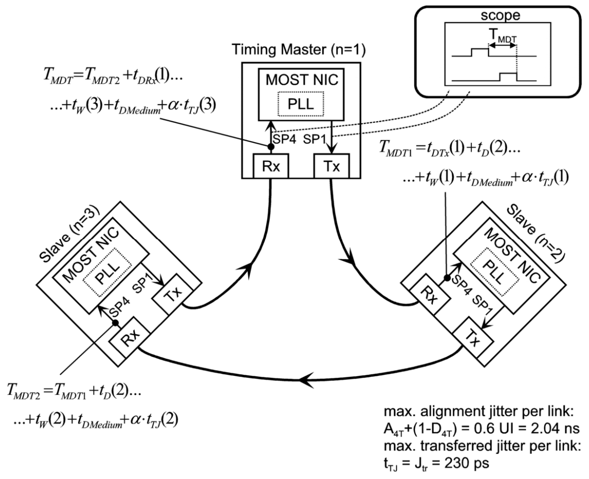

scope
T_{MDT}
Timing Master (n=1)
MOST NIC
PLL
SP4 SP1
Rx
Tx
$T_{MDT}=T_{MDT2}+t_{DRx}(1)...$
$...+t_W(3)+t_{DMedium}+\alpha\cdot t_{TJ}(3)$
$T_{MDT1}=t_{DTx}(1)+t_D(2)...$
$...+t_W(1)+t_{DMedium}+\alpha\cdot t_{TJ}(1)$
Slave (n=3)
Slave (n=2)
$T_{MDT2}=T_{MDT1}+t_D(2)...$
$...+t_W(2)+t_{DMedium}+\alpha\cdot t_{TJ}(2)$
max. alignment jitter per link:
$A_{4T}+(1-D_{4T})$ = 0.6 UI = 2.04 ns
max. transferred jitter per link:
$t_{TJ} = J_{tr}$ = 230 ps

Fig. 6.29: Position of the measuring points for the Master Delay Tolerance T_{MDT}

To measure the Master Delay Tolerance, a scope is connected to SP1 and SP4 of the master device in the ring, as indicated in Figure 6.29. The MOST150 data stream contains a period of 10 UI at the start of each frame. This long period can be used as a marker to measure the delay between any two points in the network. By capturing at least one frame, the Master Delay Tolerance can be measured. The measured value must not exceed the specified limit of 10 μs, if the sample rate is 48 kHz.

6.6 Alternative Transmission Media

After the introduction of the MOST system with PMMA fibers as transmission medium, the Physical Layer has been continuously enhanced over the last few years. The main focus of this task was on the development of alternative electrical and optical transmission media for MOST systems with higher data rates

At the present time, optical data buses on the basis of 1 mm plastic fibers and LEDs in the red wavelength range are the most cost-efficient technology for point-to-point connections in the vehicle interior. Due to higher demands on the networking in vehicles, alternative optical and electrical transmission technologies have been further developed in the last few years for future network generations. The corresponding specifications are being worked on in the MOST working group aoPhy. The following sections will introduce these technologies.

Optical Physical Bus Layer with Laser Diodes and PCS Fibers

The operational limits of the PMMA/LED-based connections are determined by the limited system margin of 13 dB, the temperature range limited to 85 °C and the limited bandwidth of ca. 100 to 200 Mbit/s. The solution to these limitations is the use of a 200 µm glass fiber with optical polymer cladding (Polymer Clad Silica, PCS), which is distinguished by a low fiber attenuation and a temperature resistance of up to 125 °C. Having a core diameter of 200 µm, the fibers are flexible enough to allow a minimum bending radius of less than 10 mm for a life cycle of 20 years in automotive surroundings. In combination with vertically emitting laser diodes (vertical cavity surface emitting laser, VCSEL) the system margin and the bandwidth can be increased.

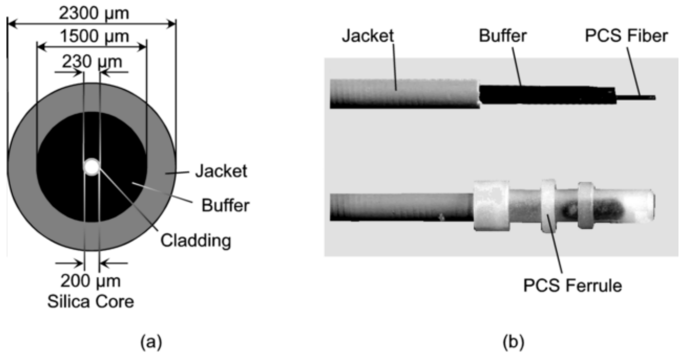


Fig. 6.30: (a): Schematic diagram of the cable structure of a PCS fiber and (b): PCS fiber with ferrule

Figure 6.30 on the left shows the cable structure of a PCS fiber for automotive applications. The fiber consists of a fused silica core with a diameter of 200 µm which is surrounded by a 15 µm optical polymer cladding. Having a core refractive index of 1.453 and a cladding refractive index of 1.405, the resulting aperture of the PCS fiber is 0.37. As in the PMMA fiber, the optical fiber is surrounded by a black buffer, in order to allow assembly of the fiber by means of laser welding. Figure 6.30 on the right shows a PCS fiber where the cladding has been removed and an assembled PCS fiber. Due to the fact that PCS fibers have a small core diameter, there are higher demands on the tolerance at the fiber contacting points. It is, however, possible to produce suitable ferrules using common injection molding procedures. Conventional cleaving procedures, where the fiber is subjected to tension and laterally scored, can be used for processing the fiber end face. This results in a flat end face with high surface quality at the point of rupture. Alternatively, the fiber end face can be processed by means of a CO_2 laser.

Figure 6.31 shows the typical attenuation spectrum of a PCS fiber. The attenuation in the wavelength range from 530 nm to 1100 nm is less than 20 dB/km and thus smaller by about an order of magnitude than the minimum attenuation of POF at 140 dB/km at 650 nm.

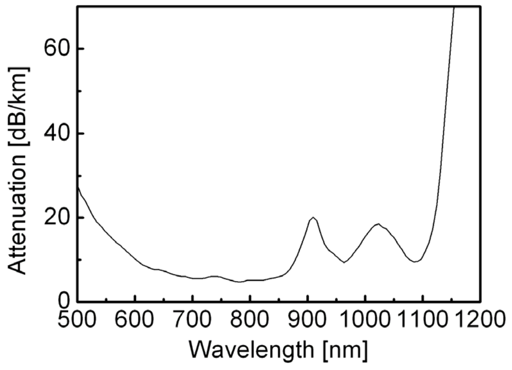


Fig. 6.31: Typical attenuation spectrum of a PCS fiber with 200 µm core diameter

Due to the attenuation minimum of PCS fibers at 850 nm, vertical emitters (VCSEL) on a GaAs basis are suited as transmitter elements. They can be modulated up to the GHz range (see fig. 6.32). In the data-com area VCSELs are widespread for short-range connections and are used for Gigabit Ethernet, for example, up to a transmission distance of more than 200 m via multi-mode optical fibers. VCSELs combine the advantages of LEDs and edge-emitting laser diodes and are distinguished by low production costs, testability at the wafer level and a simple structural design, similar to the one of LEDs. The circularly symmetrical beam profile, the small beam width and the low divergence permit a high coupling efficiency in combination with PCS fibers. VCSELs have low threshold currents, a high efficiency, and a high modulation bandwidth of far more than 1 Gbit/s and can be controlled by simple driver circuits. The reliability of VCSEL components has been proven in several studies. At temperatures of up to 125 °C, a lifetime of 10,000 hours can be reached.

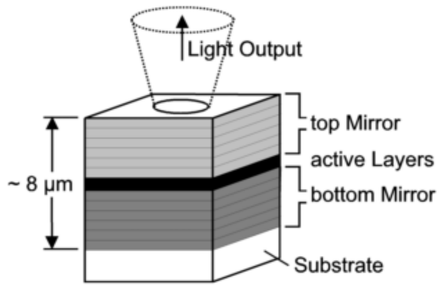


Fig. 6.32: Schematic diagram of a vertical cavity surface emitting laser diode (left) and typical characteristic curve of a VCSEL at 850 nm (right)

When a PCS fiber-based connection is used, the power budget can be increased from 13.5 dB to 18 dB compared to a PMMA fiber-based network. This is illustrated in table 6.2. By means of a temperature-compensated VCSEL, a minimum output power of -9 dBm with a simultaneous maximum output power of -1.5 dBm for the laser category 1 can be obtained. The power budget is thus increased by 1 dB compared to present PMMA connections. The sensitivity of receivers with an active surface of ca. 200 µm is at least -27 dBm, which increases the power budget by additional 4 dB. The fiber attenuation of a PCS fiber is insignificant for the fiber lengths that are commonly used.

Parameters	*PMMA/LED System*	*PCS/VCSEL System*
Max. output power	-1.5 dBm	-1.5 dBm
Min. output power	-8.5 dBm	-9 dBm
Sensitivity of receivers	-22 dB	-27 dBm
Power budget	13.5 dB	18 dB
Fiber attenuation	0.4 dB/m	0.02 dB/m
Connector interface loss	2.5 dB	2.5 dB
Inline connector interface loss	2 dB	2 dB

Table 6.2: Comparison between the power budget of a POF-LED system and a PCS-VCSEL system

Since the power budget and the temperature resistance in PCS-VCSEL systems is higher than in POF-LED systems, the existing limitations when installing the cables can be bypassed. At least two additional plug-in connectors can be inserted between two control devices in order to simplify installation of the wire harness.

A physical layer specification for the data rate of 25 Mbit/s is already available for the PCS-VCSEL technology [MOST BaPhy]. For this purpose, some manufacturers have developed prototypes of FOTs with integrated VCSEL components and photo diodes for 200 µm fibers. Figure 6.33 shows the prototype of a transmitter with integrated VCSEL and laser driver. The components are encapsulated in a plastic housing to implement a simple and low-cost package. The figure also shows different versions of MOST transceivers of the 2+0, 2+4 and 2+40 type.

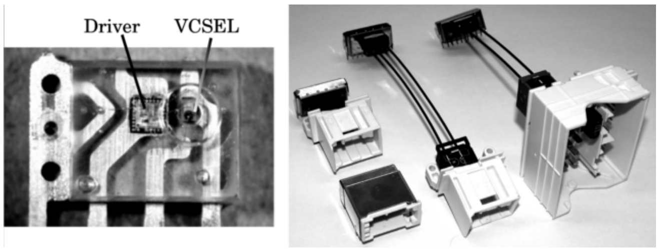


Fig. 6.33: VCSEL transmitter and driver in injection-molded housing (source: Finisar Advanced Optical Components Division) and modified MOST transceiver with VCSEL transmitter for PCS fibers (source: Yazaki Corporation)

At present, PCS-VCSEL systems are not yet being used in vehicles due to higher mechanical tolerance demands on the components and resulting higher system costs. This technology will only become commonly used when the demands on data rates or the power budget increase to such an extent that the LED-POF technology comes up against its physical limits.

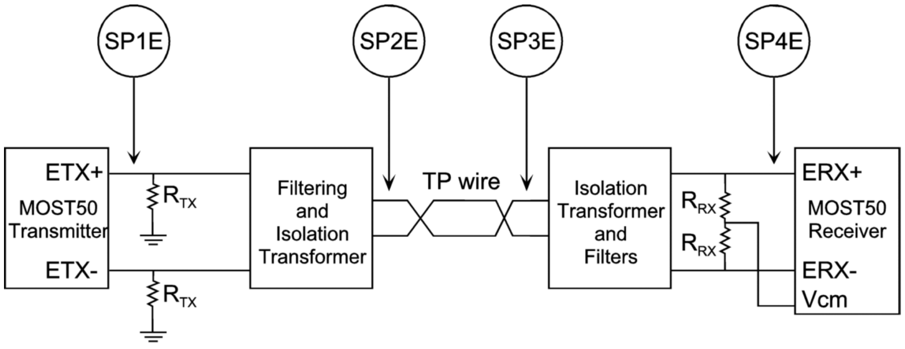


Fig. 6.34: Schematic diagram of an electrical MOST connection

Electrical Physical Bus Layer at 50 Mbit/s

As an alternative solution to optical systems, electrical transmission technologies for MOST at a data rate of 50 Mbit/s are presently being developed. The motivation for these endeavors is the desire to use existing electrical wiring technologies and to save costs as compared to optical connections. Due to the lower electro-magnetic compatibility, however, electrical data lines must be shielded. Otherwise, complex driver circuits are required to meet the respective EMC demands. Both measures, in turn, increase the system costs. The advantage of lower costs thus mainly depends on the EMC requirements or the expense for EMC protection for "worst case" conditions.

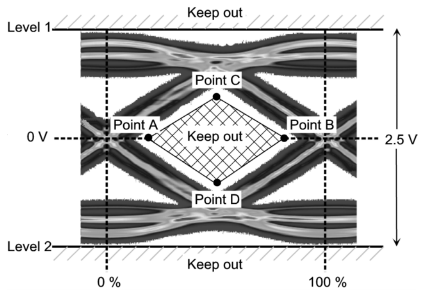


Fig. 6.35: Eye diagram of an electrical MOST signal with test mask

The physical bus layer for electrical transmission consists of a twisted-pair wire for transmitting a differential signal in combination with suitable transmitters (or transformers) forming the interface between transmission medium and MOST Network Interface Controller. Following the specification of the optical physical bus layer, the specification points of the electrical physical bus layer are referred to as SP1E, SP2E, SP3E and SP4E (see fig. 6.34).

The electrical specification parameters are basically predetermined by the signal amplitude and the common mode voltage at a given load resistance and load capacitance. In addition, the electrical signals, as in the optical transmission technology, must meet specified timing requirements (maximum pulse width distortion and phase variations). The electrical signals can be evaluated using an eye diagram. Figure 6.35 shows an example of an eye diagram of an electrical signal and the pattern of a test mask. The electrical Physical Layer Specification [MOST ePhy] specifies a particular test mask and the parameters level 1, level 2 and the points A to D of the trapezoid for all interfaces SP1E, SP2E, SP3E, and SP4E.

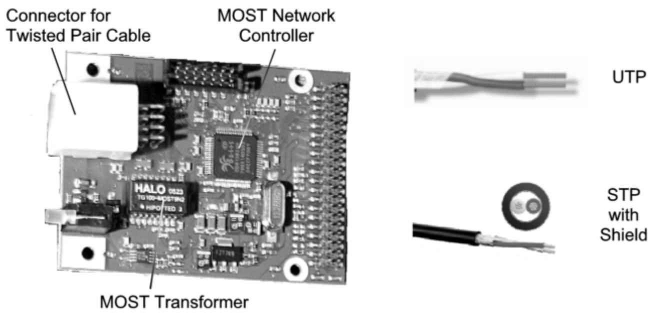


Fig. 6.36: Test circuit board with transmitter (left, source: SMSC Europe) and comparison between shielded and unshielded electrical transmission media (right)

Fig. 6.36 shows on the left an example of a test circuit board with a network controller (NIC), a MOST transformer and a connection socket for transmitters (TX+/TX-), receivers (RX+/RX-) and connections for the shielding. On the right, different electrical transmission media are shown. Unshielded (unshielded twisted pair, UTP) and shielded (shielded twisted pair, STP) wires can be used for connection. It depends on the desired line length, the EMC requirements and the transmission properties of the used transformers, which suitable medium is selected.

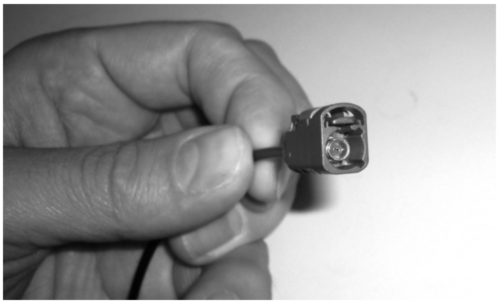


Fig. 6.37: Example of a coax cable and coax connector

Electrical Physical Bus Layer, based on Coax Line Drivers

A new approach for the electrical physical layer is to transmit the data over coax cables in combination with coax line drivers. The idea is to replace the opto-electronic converter by a line driver and the use of standard connector and standard coax cable that are well known and established in the automotive area. Fig. 6.37 shows an example of the cable and connector.

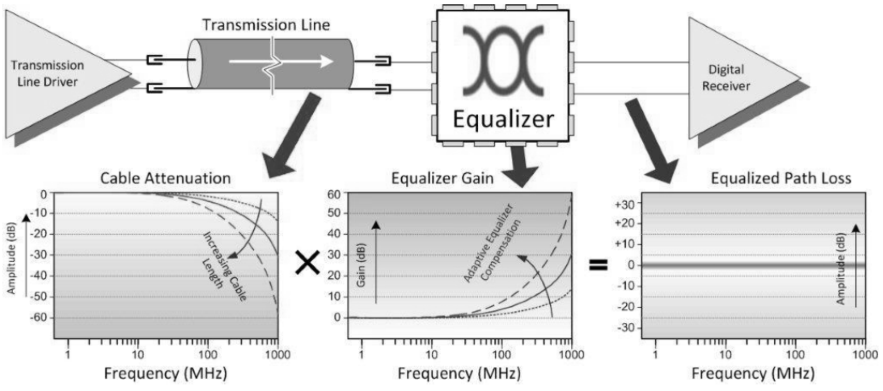


Fig. 6.38: Schematic of an electrical transmission line, indicating the compensation of the cable attenuation by an equalizer (source: Eqcologic)

The main innovations of the system are the possibility to transmit in both directions and to provide the power supply with the same cable. This gives MOST the opportunity to expand in other domains like in assistance systems where cameras have to be integrated with small connectors due to limited installation space. To overcome the bandwidth limitation of coax cables, it is essential to use an equalizer at the receiver side (see fig. 6.38). With an equalizer, the attenuation in the high frequency domain of the cable is compensated. It is useful to regulate the gain by an adaptive equalizer to compensate variations of the cable attenuation due to different cable quality and length. The physical layer working group is presently specifying this technology and will release both, an adapted basic specification and a new Electrical Physical Layer Sub-Specification. The system requirements concerning EMI and immunity against different ground levels are also considered in this context.

7 Network and Fault Management

The MOST System comprises several master functionalities, each of which is only allowed once in the system. They were already discussed in chapter 2. The following chapter goes into detail regarding the NetworkMaster – responsible for the network management. The NetworkMaster is the device which contains the function block *NetworkMaster*. All other devices in the MOST network are referred to as Slave devices or Slave components. The system is initialized by the Slave components exchanging information with the NetworkMaster during the start-up.

7.1 System Initialization

After wake-up, the NetworkMaster first of all builds up the communicative relations to the Slave components via a system scan. After successful system initialization, the communicative relations between the individual Slave components are established.

7.1.1 System Query

If there is a stable lock, the NetworkMaster starts querying all nodes present about their function blocks. It addresses each node position address. The Network and the TimingMaster reside in the same node. The TimingMaster is always the node zero in the ring, i.e., it has the node position address 0x400 and the logical address 0x100.

The queried node informs the NetworkMaster about its stored logical node address and the function blocks ready for communication:

```
0x0100→0x0401: NetBlock.01.FBlockIDs.Get ( )
0x0101→0x0100: NetBlock.01.FBlockIDs.Status (FBlockIDList)
```

The NetworkMaster can read the information about the number of nodes in the system from the maximum position register (MPR) of its NIC. From the answer, the NetworkMaster can extract the following information about the queried node:

- Node position and logical address of the node (more details in section 5.7)
- Function blocks, which are supported by the node (FBlockIDList)
- InstanceID of the respective function block

With the NetBlock (FBlockID 0x01), each MOST node has at least one FBlock; usually, several additional application FBlocks are implemented. The NetBlock is, however, not transmitted in the FBlockIDList.

7.1.2 Central Registry

The NetworkMaster stores the information obtained by querying the node in the *Central Registry.* It is filled with the logical addresses and the corresponding function blocks of all nodes; not, however, with the node position address. The Central Registry is thus a logical map of the system and persists to the end of the system run. Figure 7.1 shows the query sequence for filling the Central Registry.

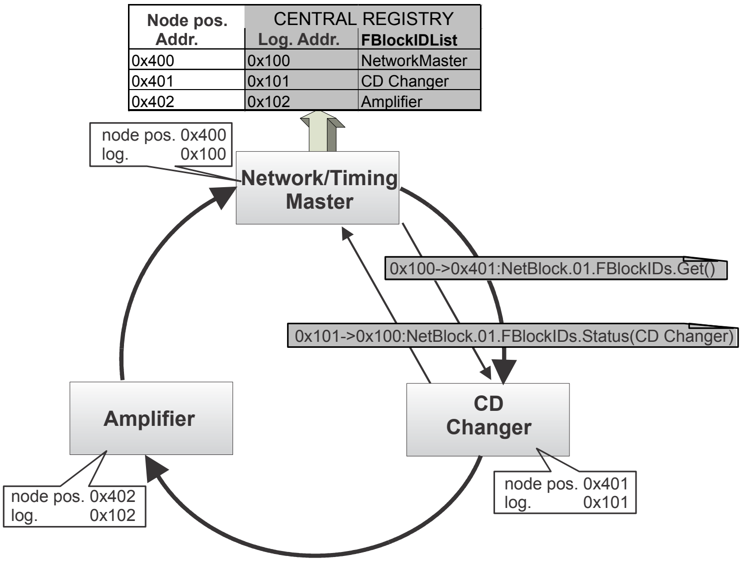


Fig. 7.1: Illustration of the query sequence and filling of the Central Registry

As soon as valid System State OK is entered, the stored information is available for all participants in the ring. The NetworkMaster can then compare with a previously stored registry and detect possible changes of the current network configuration. A stored registry can possibly be a stored Central Registry of the last system run or a registry pre-specified at the car assembly end of line test.

Network Status

The System State is distributed throughout the system by the NetworkMaster on the protocol level by means of the message

```
NetworkMaster.01.Configuration.Status
```

It indicates the current status of the system at any time and is sent as a broadcast message to all nodes in the MOST system by the NetworkMaster.

System State OK

The Central Registry is valid, the NetworkMaster sends the message `Configuration.Status(Ok)`. The reception of this status message means for all nodes that the Central Registry can be queried. The Slave components can then establish their communicative relations with each other.

System State NotOK

If the current Central Registry does not match a stored registry, or if there is a system conflict, all communicative relations are aborted using the message `Configuration.Status(NotOK)` by the NetworkMaster and a system scan is initiated (see section 7.1.4).

The System State remains at NotOK until an OK message is sent off.

Configuration.Status (NewExt)

The notification of the function block signalizes the readiness of the underlying application to communicate. In some cases applications require more start-up time to avoid confusion with notification. This is mostly the case with complex applications.

A node that wants to register additional new function blocks after the `Configuration.Status(OK)` was sent, must do so separately by way of the following message:

```
0x1xx→0x100 NetBlock.xx.FBlockIDs.Status (FBlockIDList)
```

The NetworkMaster compares the newly registered function blocks to the ones stored in its Central Registry. If new function blocks were added, this is broadcast throughout the system via the message `Configuration.Status(NewExt, FBlockIDList)`.

This message can only be sent in the OK System State. The System State remains in OK.

The case is different if there is an additional node participating in the system. The NetworkMaster then queries all nodes about their function blocks again. It can thus identify the new node and ascertain the node address. This is the big advantage of dynamic address allocation (see section 7.1.4). If a new node is added in the middle of the ring, there is no system conflict if the new node has a free and unused logical address.

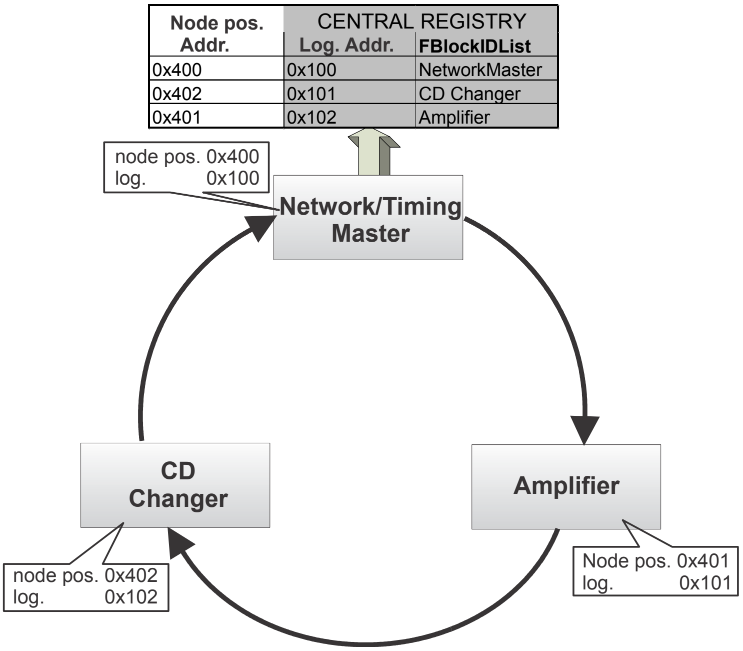
Configuration.Status (Invalid)

Apart from late registration of function blocks, a device can also deregister its application or function block. There might also be an unexpected problem within the device so that the entire node is no longer available in the system. For example, a MOST device must close its electronic bypass for a certain period after an internal reset.

If there is a change in the system to the effect that a node is no longer available (e.g. due to a reset), the NetworkMaster starts querying all nodes in the system about their function blocks. By comparing the result with the Central Registry, it can easily detect the dropped-out node. This is then broadcast throughout the system using the message:
`Configuration.Status (Invalid, FBlockIDList).`

This message can only be sent in the OK System State. The System State remains in OK.

After function blocks have dropped out, the communication partners of the function block concerned must update their Notification Matrix (see section 7.2) accordingly, i.e., they must cancel the dropped out function block. The same applies to the Decentral Registry, which is discussed in detail in section 7.2.



Node pos. Addr.	CENTRAL REGISTRY	
	Log. Addr.	FBlockIDList
0x400	0x100	NetworkMaster
0x402	0x101	CD Changer
0x401	0x102	Amplifier

Fig. 7.2: Using the logical address for communicative relations. The Central Registry has not changed compared to fig. 7.1.

7.1.3 Dynamic Addressing

The Central Registry only stores the logical, not the node position addresses. Both types of addresses are decoupled from each other. The nodes get information via the Central Registry about the addresses of function blocks that are found, in order to establish direct communicative partnerships with each other. As only the logical address is used for the direct communication between the Slave nodes, there is no correlation with the actual node positions in the ring. If a node is added or dropped between the nodes of two communication partners, the logical address remains unchanged, whereas the node position address of all following nodes is changed. The communicative relation thus remains unchanged.

If the NetworkMaster receives registrations from a Slave node of the address 0xFFFF during the system run (not only during the initialization of the system), there is a system conflict. The NetworkMaster aborts all other actions and broadcasts `Configuration.Status(NotOK)`. This causes all Slave nodes to discard their stored logical address and to recalculate it according to

RxTxLog = 0x100 + Pos

7.1.4 Examples of System Starts

The following configuration is selected for the examples:

	Node Position Address	*Logical Address*	*Function Block*	*FBlock-ID*
Node 0	0x400	0x100	NetworkMaster	0x01
Node 1	0x401	0x101	AudioDiskPlayer	0x31
Node 2	0x402	0x102	AudioAmplifier	0x22

1. The Central Registry did not change after the latest system run. The AudioDisk Player registers late.

```
0x0100→0x0401: NetBlock.01.FBlockIDs.Get ()
0x0101→0x0100: NetBlock.01.FBlockIDs.Status ()
0x0100→0x0402: NetBlock.02.FBlockIDs.Get ()
0x0102→0x0100: NetBlock.02.FBlockIDs.Status (0x22,0x00)
0x0100→0x03C8: NetworkMaster.01.Configuration.Status (OK)
0x0101→0x0100: NetBlock.01FBlockIDs.Status (0x31,0x00)
0x0100→0x03C8: NetworkMaster.01.Configuration.Status(NewExt,
                  0x22,0x00)
```

2. The AudioAmplifier was separated from the power supply for a considerable period of time. It lost its logical address.

```
0x0100→0x0401: NetBlock.01.FBlockIDs.Get ()
0x0101→0x0100: NetBlock.01.FBlockIDs.Status (0x31,0x00)
0x0100→0x0402: NetBlock.02.FBlockIDs.Get ()
0xFFFF→0x0100: NetBlock.02.FBlockIDs.Status (0x22,0x00)
0x0100→0x03C8: NetworkMaster.01.Configuration.Status (NotOK)
0x0100→0x0401: NetBlock.01.FBlockIDs.Get ()
0x0101→0x0100: NetBlock.01.FBlockIDs.Status (0x31,0x00)
0x0100→0x0402: NetBlock.02.FBlockIDs.Get ()
0x0102→0x0100: NetBlock.02.FBlockIDs.Status (0x22,0x00)
0x0100→0x03C8: NetworkMaster.01.Configuration.Status (OK)
```

7.2 Notification

Decentral Registry

If the NetworkMaster sends out a `Configuration.Status(OK)`, establishment of communication between the Slave devices is started off. This is usually effected by obtaining the logical address for each interesting function block from the components at the NetworkMaster.

```
0x0101→0x0100: NetworkMaster.01.CentralRegistry.Get
                 (FBlockID, InstID)
0x0100→0x0101: NetworkMaster.01.CentralRegistry.Status
                 (FBlockInfoList)
```

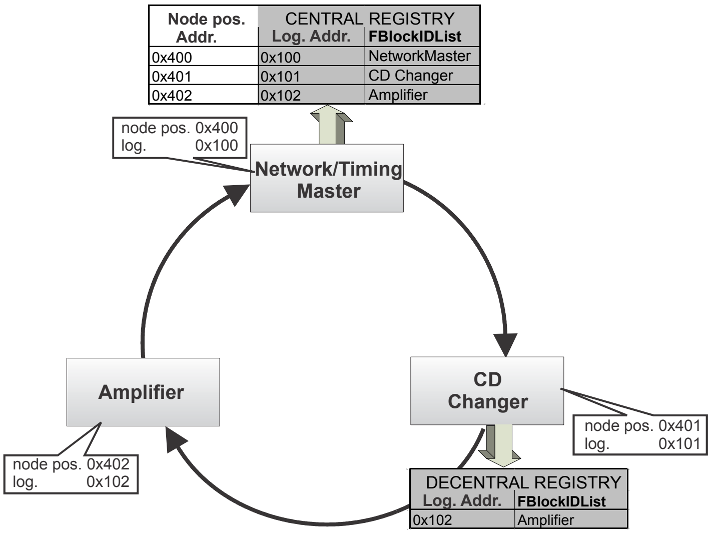


Fig. 7.3: Configuration of the Decentral Registry from information of the Central Registry

This information is stored in the Decentral Registry, as it is called, in the respective slave. The Decentral Registry facilitates the finding of communication partners, as it locally stores the result of each search. When accessing the same communication partner again, the Central Registry does not need to be contacted again. It remembers the communication partners. The Decentral Registry and the Central Registry may use the same syntax. Only the content is reduced to the information necessary for the communication of the partner device. Figure 7.3 shows the sequence for the configuration of the Decentral Registry. With the help of the Decentral Registry the device can then communicate directly with the communication partner.

Notification Mechanism

In many cases devices must be informed of property changes in function blocks belonging to other devices. If this had to take place by a poll procedure, there would be a heavy load on the bus. In order to prevent this, the mechanism of notification was created. In the case of a property change, an event is sent to the devices that are entered in the Notification Matrix. The Notification Matrix is a table, which is implemented in the Network Service Layer II (section 9.3.3).

```
0x0101→0x0100: FBlock.InstID.Notification.Set (Control,
                DeviceID, FktID1, FktID2,…)
```

	FktID1	*FktID2*	*FktID3*
DeviceID1	x	x	
DeviceID2		x	x
DeviceID3	x		x
free			
free			

Example

A display unit (DeviceID 0x100) shall always display the current track number of an AudioDiskPlayer. For this purpose, it sets notification for the function TrackPosition (FktID 0x202) of the AudioDiskPlayer and is automatically informed of the new position at each track change. The following list shows the required control messages. For further details, refer to the MOST Specification [MOST 3.0].

```
0x0100→0x0101: AudioDiskPlayer.01.Notification.Set (0x1,
0x100, 0x202)
0x0101→0x0100: AudioDiskPlayer.01.TrackPosition.Status (01)
0x0101→0x0100: AudioDiskPlayer.01.TrackPosition.Status (02)
:
0x0101→0x0100: AudioDiskPlayer.01.TrackPosition.Status (05)
:
```

7.3 Power-Down

The shutdown of a MOST system is basically divided into two phases. In a preparatory phase, the system participants are notified of the shutdown. In the second phase, the system is actually shut down and proceeds to sleep mode. The phases are referred to as *initiation* and *execution* of the shutdown. As in the case of the start-up, the PowerMaster takes on the control function for the power-down.

Initiation of the Shutdown

Initiation of a shutdown is usually effected by an external event, such as locking the door of the vehicle. This trigger event is transmitted to the PowerMaster by way of a MOST message. The PowerMaster then starts initiating the shutdown with the following broadcast message:

```
0x0100→0x03C8: NetBlock.00.ShutDown.Start (Query)
```

The system participants can then prepare for the sleep mode. If a device requires ongoing communication and is not yet ready to fall asleep, it informs the PowerMaster:

```
0x0101→0x0100: NetBlock.01.ShutDown.Result(Suspend)
```

The PowerMaster waits for a certain period of time $t_{WaitSuspend}$ and then repeats its query.

Execution of the Shutdown

If the PowerMaster did not receive any objections from the devices within the time $t_{WaitSuspend}$, it executes the shutdown by way of the following message:

```
0x0100→0x03C8: NetBlock.00.ShutDown.Start (Execute)
```

After that, the PowerMaster switches off its modulated signal within the time $t_{ShutdownWait}$ (where $t_{ShutdownWait} = 2$ sec usually). All subsequent nodes also switch off their modulated signal, as soon as they have not received a light from the RX input.

7.4 Fault Types

This section differentiates in two kinds of faults: Network faults and device faults.

7.4.1 Network Faults

Ring Break

Normally, a device in a MOST system is woken by the reception of a modulated signal at its RX input. As a consequence, it transmits a signal on its TX output. If a TimingMaster transmits a signal on its TX output, it usually receives a signal on its RX input within a defined time if all TimingSlaves are connected in a ring topology. However, it is possible that the ring cannot be locked because of an inaccurately connected component, a defect component or a broken cable. In this case, a start-up for normal operation is not possible anymore.

Sudden Signal Off

If a device loses the signal at its RX input abruptly (i.e., without a previously initiated shutdown procedure), it automatically switches to TimingMaster mode. Afterwards, it sets the Shutdown Flag in the administrative area of the transmitted MOST frames in order to inform the other nodes about the coming shutdown. After transmitting this signal within the time interval $t_{SSO\_ShutDown}$, it turns off the transmission and stores the cause of the fault. Because the other nodes have detected the Shutdown Flag, they do not store a fault event before also shutting down. Hence, a Sudden Signal Off leads to a shutdown of the whole MOST network.

Unlock and Critical Unlock

The loss of synchronization between the system clock of the TimingMaster and a TimingSlave device is referred to as *unlock*. An unlock can have many reasons, such as a faulty RX or TX component or insufficient signal intensity for detecting the system clock. A differentiation is made between short and critical unlocks. A short unlock is defined by the loss of synchronization for less than t_{Unlock} (usually less than 100 ms, typically 70 ms). A short unlock is compensated by the TimingMaster regenerating the signal.

There is a critical unlock if the loss of synchronization takes longer than t_{Unlock}. According to the behavior described in the paragraph "Sudden Signal Off", the affected device automatically switches to TimingMaster mode, transmits MOST frames with an appropriate Shutdown Flag, stores the fault reason, and finally shuts down.

Coding Errors

Coding errors have their reason in a violation of the coding scheme, e.g., due to high signal attenuation based on a deformed cable. Because the nodes cannot decode the received data, there is either delayed communication with many retransmissions, or even data loss. A massive occurrence of Coding errors can lead to an Unlock, a Critical Unlock or a Sudden Signal Off.

7.4.2 Device Faults caused by Temperature and Voltage Violations

This section considers faults that occur if MOST devices are shut down to protect them against malfunctions or permanent damage caused by temperature and voltage violations.

Of course, in a system further device faults can occur. However, faults caused by temperature and voltage violations are of particular importance for a MOST network because the sudden shut down of a MOST device leads to a Sudden Signal Off event. MOST provides special mechanisms that allow differentiating in Over-Temperature, Undervoltage and Overvoltage faults and special handling of them.

Over-Temperature

High demands are made on the vehicular devices as far as the temperature range is concerned. Due to physical circumstances, the required operational temperature cannot always be observed. Each participant in the ring therefore has the possibility to initiate a temperature-related shutdown to protect all components. If a device exceeds its specified temperature $\vartheta_{Shutdown}$, it broadcasts the message `NetBlock.ShutDown.Result (Temperature Shutdown)`. If the critical temperature $\vartheta_{Critical}$ is reached, the device initiates an immediate shutdown by simply switching the modulated signal off. Because a PowerMaster has previously received the temperature shutdown request, it waits a certain time for cooling down and afterwards it wakes up the system again.

In addition, a further kind of the temperature shutdown can be used: if the temperature ϑ_{AppOff} is reached, it is optional to shut down only a part of the application (for example an overheated drive). The remaining functionality can still be used. The corresponding FBlock must be unregistered from the Central Registry.

Undervoltage

Undervoltage in the vehicle power system does not usually affect all devices at the same time and to the same extent. There are thus two different undervoltage thresholds:

- Critical Voltage $U_{Critical}$: This undervoltage level is defined such that network communication is still ensured, but not the complete function of an application. The Network Service remains in the *Normal Operation* mode. Data is not sent over streaming connections anymore. It means, a source must route zeros and a sink must secure its output signals. The Mute property remains unchanged. When the normal voltage has been reached, the data can be sent again.

- Low Voltage U_{Low}: This undervoltage level is defined such that communication of the network interfaces can no longer be ensured. If U_{Low} is reached, the device switches off the modulated signal and switches to DevicePowerOff mode. The device stays in DevicePowerOff mode, even if the supply voltage recovers. It is awakened either by "modulated signal on" at its input or by the demand for communication from its own application. It changes to mode DeviceNormalOperation via the standard initialization process.

Overvoltage or Super voltage

If the voltage exceeds super voltage (U_{Super}), the device has to move in a safe operation state, which must be defined for each device individually. A typical value for USuper is 16V.

8 Network Diagnostics

In this chapter, the handling of diagnostics in a MOST system is described. It considers mechanisms for identifying network faults and for localizing their origins in the network. These mechanisms are important especially in the automotive environment because they reduce costs during system production and maintenance. Finally, the chapter considers faults that occur in MOST devices. Using a special communication protocol (Diagnostics Adaptation Protocol, DAP) a diagnostics test tool is able to read out the fault memories of MOST devices.

8.1 Network Fault Detection

MOST is able to handle several network fault types.

8.1.1 Ring Break Diagnosis

As a result of the ring break diagnosis (RBD), every device stores its node position in relation to a ring break. This is possible if the device behind the ring break becomes the TimingMaster. It transmits a signal that contains the node position. Each TimingSlave increments the received node position value. If no ring break is detected, the original TimingMaster begins normal operation.

At first, an external trigger is applied to the device for starting the RBD. The trigger event is system specific. For instance, it is possible that a system test on the Electrical Control Line (see also section 8.1.4) executes the RBD.

Figure 8.1 shows exemplarily the RBD procedure.

The original TimingMaster of the example network is Node 0. Node 1, Node 2, and Node 3 originally act as TimingSlaves. There is a ring break between Node 1 and Node 2. At the beginning, each TimingSlave waits for activity on its RX input. A node switches from TimingSlave mode to TimingMaster mode if it has not detected any activity during the time interval t_{Diag_Signal}. Therefore, Node 2 and Node 3 act as TimingMaster after t_{Diag_Signal}. Afterwards, Node 3 receives a signal from Node 2. Therefore, Node 3 switches immediately back to the TimingSlave mode. After the time interval T1 the original TimingMaster, Node 0, evaluates its RX input. Because it detects a signal but no stable lock, it recognizes that there is a second TimingMaster in front of it. Therefore, it switches from TimingMaster mode to TimingSlave mode. This procedure leads to an open chain beginning with a TimingMaster (Node 2) and continuing with all other nodes as TimingSlaves (Node 3, Node 0, and Node 1). The TimingMaster Node 2 transmits a signal containing the node position 0. Node 3 receives this signal and increments the node position. It stores node position 1, and transmits a signal containing this new node position value. The node position handling is continued by the following TimingSlaves.

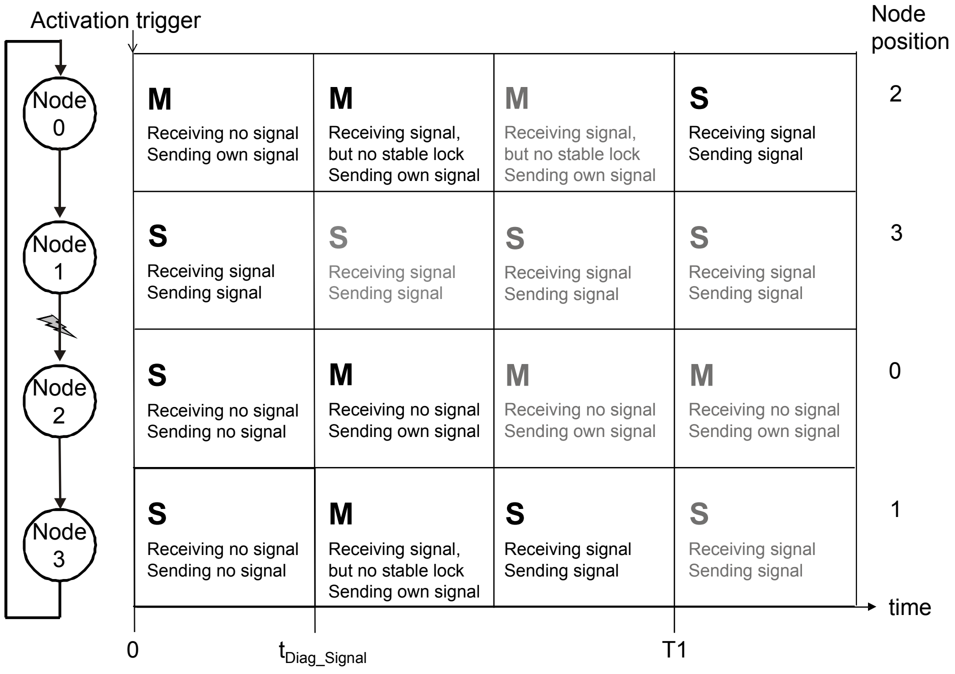

Activation trigger
Node position
Node 0
Node 1
Node 2
Node 3
M Receiving no signal Sending own signal
M Receiving signal, but no stable lock Sending own signal
M Receiving signal, but no stable lock Sending own signal
S Receiving signal Sending signal
2
S Receiving signal Sending signal
S Receiving signal Sending signal
S Receiving signal Sending signal
S Receiving signal Sending signal
3
S Receiving no signal Sending no signal
M Receiving no signal Sending own signal
M Receiving no signal Sending own signal
M Receiving no signal Sending own signal
0
S Receiving no signal Sending no signal
M Receiving signal, but no stable lock Sending own signal
S Receiving signal Sending signal
S Receiving signal Sending signal
1
time
0
t_Diag_Signal
T1

Fig. 8.1: Localization of the ring break

Finally, it is possible that the RBD result is delivered to a dedicated device. This last step is optional and depends on the system solution.

8.1.2 Sudden Signal Off/Critical Unlock Detection

This function is new in the MOST Specification Rev. 3.0. If a node causes a Sudden Signal Off (SSO) or a Critical Unlock (CU) event in NetInterface Normal Operation mode, the succeeding node cannot receive a valid signal anymore. As a consequence, this succeeding node switches to TimingMaster mode, sets the Shutdown Flag in the administrative area of the transmitted MOST frames, stores the fault cause and finally shuts down as already described in section 7.1.

The original TimingMaster cannot detect the Shutdown Flag. Therefore, it also detects an SSO or a CU fault and stores it. This must be considered in an appropriate fault evaluation.

An evaluating device can use the MOST property `NetBlock.ShutDownReason` to collect the locally stored fault information. When it has gathered the information from all nodes, it broadcasts `NetBlock.ShutDownReason.Set(0x00)`. Afterwards, all nodes delete the stored shutdown reason.

An evaluating device must apply an evaluation algorithm in order to detect the origin of the fault because it is possible that more than one node reports a fault:

- If a TimingSlave reports an SSO, an SSO fault is definitely in front of the TimingSlave. Because of fault propagation, it is possible that other nodes report CUs. The evaluation algorithm should ignore them.
- If the TimingMaster reports an SSO and no TimingSlave reports an SSO or CU, the SSO fault is definitely in front of the TimingMaster.
- In principle, there could be more than one CU report because of fault propagation in the MOST network. If no additional SSO is reported, the initial CU is in front of the first TimingSlave that has reported the CU.
- If at least one TimingSlave reports an SSO or CU, the situation in front of the TimingMaster is unclear. Therefore, the evaluation algorithm should ignore the report of the TimingMaster.

8.1.3 Localizing the Origin of Coding Errors

The Network Service of each MOST node is able to count coding errors. They are usually not counted during start-up or shutdown of the network but while the system is normally operating. Furthermore, a node does not count coding errors if unlocks, CUs, or SSOs occur. A node counts at maximum one coding error within a system specific time interval.

A dedicated node can use the MOST property `Diagnosis.CodingError` to read the coding error counter of each node. It detects the origin of a coding error fault directly in front of the node with the lowest node position that reports a number of coding errors that is greater than a system specific threshold. The evaluating node should ignore the coding error reports of succeeding nodes to compensate for fault propagation.

8.1.4 Electrical Control Line

Some car manufacturers have decided to introduce the ECL (Electrical Control Line) in addition to the MOST network in order to be able to detect network faults even if the faulty MOST network prevents any communication. This new optional feature is based on MOST Specification Rev. 3.0 and described in a separate document [ECL]. The ECL connects the MOST devices by additional cabling. These connected devices are able to exchange information in an electrical, serial and digital way via the ECL.

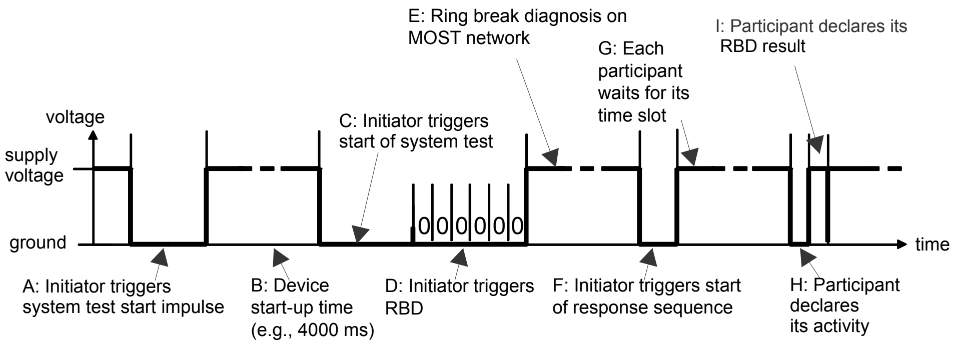

E: Ring break diagnosis on MOST network
I: Participant declares its RBD result
G: Each participant waits for its time slot
voltage
C: Initiator triggers start of system test
supply voltage
0 0 0 0 0 0
ground
time
A: Initiator triggers system test start impulse
B: Device start-up time (e.g., 4000 ms)
D: Initiator triggers RBD
F: Initiator triggers start of response sequence
H: Participant declares its activity

Fig. 8.2: Sequence of a system test

The idea is that a dedicated device uses the ECL in order to initiate the execution of a so-called system test. A system test can include ring break diagnosis, SSO/CU fault origin detection, or coding error counting. After finishing a system test, each participating device sends its test result back to the initiator via the ECL.

Figure 8.2 depicts exemplarily the whole sequence of a system test.

In step A, the initiator device transmits a system test start impulse to all participating devices by pulling the voltage of the ECL down to ground for 200 ms. During step B, there is no activity on the ECL. All participating devices have time to start up. In step C, the initiator synchronizes all participating devices by transmitting an impulse that identifies the start of the system test. To this end, it pulls the voltage down to ground for 250 ms. At step D, the initiator triggers a system test by sending an appropriate parameter sequence. Let the value “0” be defined by pulling voltage to ground for 50 ms, and let value “1” be defined by leaving the ECL at supply voltage for 50 ms, then the parameter sequence

- “0-0-0-0-0-0”defines an RBD test. If a participant detects a stable lock, it sends “0” in its result time slot at step I; else, it sends “1”.
- “0-1-0-0-0-0” defines a coding errors test. If the detected number of coding errors exceeds a system specific threshold, the participant sends “0” at step I; else, it sends “1”.
- “1-0-0-0-0-0” defines an *alive* test. As shown in figure 8.2, there is a time slot at step H for each participant to declare its general activity. For each system test, an active participant sends “0” in this time slot. In case of an alive test, the participant sends no further result at step I. An alive test can be applied for checking the general availability of the devices.

- “1-1-0-0-0-0” defines an SSO/CU test. If a participant does not detect an SSO or CU during the test, it sends “0” at step I; else, it sends “1”.

While executing the system test, all devices leave the ECL at supply voltage (see step E). At step F, the initiator triggers the start of the response sequence by pulling the voltage down to ground for 100 ms. This activity synchronizes all participating devices. Afterwards, each participant must wait for its time slot (step G). Each device has been statically assigned its own time slot. This assignment is independent of the MOST ring position. As already mentioned, a participant responds at step H and step I.

8.2 Device Fault Diagnosis

This section introduces the Diagnostic Adaptation Protocol (DAP). It adapts a communication protocol for vehicular diagnosis to the actual communication protocols of MOST. This function is new in the MOST Specification Rev. 3.0; its use is optional and is described in a separate document [DPA].

The basic concept for vehicle diagnosis is this: Diagnostic services reside in devices in vehicles. A diagnostic test tool acts as client by requesting these services. This way, a diagnostic test tool in a workshop can read, e.g., the trouble code of each device. Every fault occurrence in a device leads to a new trouble code entry in its fault memory. A trouble code entails an appropriate instruction for an action by the workshop staff.

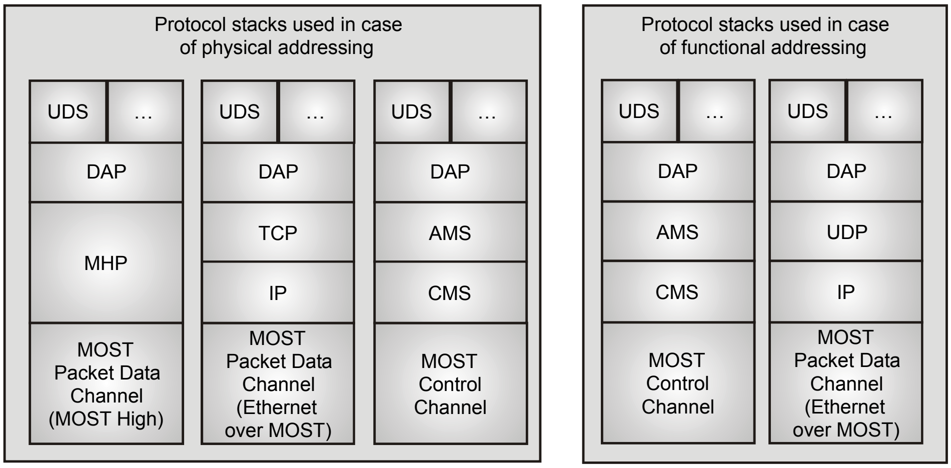

Protocol stacks used in case of physical addressing
UDS
…
DAP
MHP
MOST Packet Data Channel (MOST High)
UDS
…
DAP
TCP
IP
MOST Packet Data Channel (Ethernet over MOST)
UDS
…
DAP
AMS
CMS
MOST Control Channel
Protocol stacks used in case of functional addressing
UDS
…
DAP
AMS
CMS
MOST Control Channel
UDS
…
DAP
UDP
IP
MOST Packet Data Channel (Ethernet over MOST)

Fig. 8.3: Protocol Stacks

For realizing the client/server communication, many protocols can be applied. For instance, Unified Diagnostic Services (UDS) [ISO 14229] is often used for the purpose of vehicular diagnosis.

Figure 8.3 shows the communication protocol stacks that realize a diagnostic message exchange on MOST using DAP.

Physical addressing means that the client sends requests to exactly one server. It is possible that after service execution the server sends back a final response. For instance, a response can contain the requested diagnostic trouble code of a device. Furthermore, the server may send back *response pending* messages while executing a diagnostic service. Physically addressed requests and their corresponding responses can be transported via the MOST Packet Data Channel using MHP or TCP, or via the MOST Control Channel using AMS.

Functional addressing means that the client sends requests to many distributed servers. Multicast or broadcast mechanisms are the obvious means of transporting these requests. A server has the possibility to send back *response pending* messages or a final response message. Functionally addressed requests can be transported via the MOST Packet Data Channel using UDP, or via the MOST Control Channel using AMS. The usage of either blocking or unblocking broadcast as specified in the MOST Specification [MOST 3.0] is possible.

Byte Order	*Data Field*	*Len (Byte)*	*Value*
1	Version	1	Version of DAP Header: 0x01
2 – 5	Payload Length	4	n
6	Type of Service	1	System Integrator specific: 0x00 UDS: 0x01 KWP2000: 0x02 Reserved for future use: 0x03 – 0xBF System Integrator specific: 0xC0 – 0xFF
7 - 8	System Integrator Part	2	System Integrator specific: 0x0000 – 0xFFFE Default: 0xFFFF
9 – 10	Source Address	2	e.g., Diagnostic test tool address Not used: 0xFFFF
11 – (10+n)	Payload	n	

Table 8.1: Structure of a DAP packet

The first data field identifies the version of the DAP header, which currently is 0x01.

Afterwards there is the data field “Payload Length”. For stream-oriented TCP, this field can be used for reassembling a diagnostic message from the byte stream. Furthermore, if the UDS message size is greater than the maximum MHP packet size, this field is needed for segmentation and reassembly, too.

The next data field “Type of Service” selects the application protocol. The payload of a received DAP packet can be assigned to UDS, or to another application protocol (e.g., KWP2000).

The data field “System Integrator Part” is set to 0xFFFF by default. It can be used by the System Integrator, e.g., in order to work with sequence numbers if needed.

The data field “Source Address” identifies the diagnostic test tool that has sent a request. A server device has to copy the data of this field from a diagnostic request to the corresponding diagnostic response. Thus, the transport of a diagnostic response to the original diagnostic test tool is possible even when there is more than one diagnostic test tool and if the response must be transported via more than one network.

Finally, the data field “Payload” contains the diagnostic message that is processed by UDS, or by another application protocol (e.g., KWP2000).

9 Network Service

9.1 Overview

The Network Service represents the standard protocol stack for MOST. The commonly used term *MOST NetServices* is a trade name of SMSC Corporation, where the *MOST NetServices* have to be licensed. The MOST Specification uses the term *Network Service*, however, as a term which is independent of a product.

As shown in figure 9.1, the Network Service is based on the Network Interface Controller (NIC) or the Intelligent Network Interface Controller (INIC) and provides a programming interface for the application which basically consists of the function blocks. It comprises modules for transferring packet data in the Packet Data Channel and modules for controlling the network via the Control Channel, i.e., it contains mechanisms and routines for operating and managing the network and it ensures dynamic behavior of the network. The Network Service is implemented on the External Host Controller (EHC). Control of the Streaming Data Channels is also part of the Network Service; the transmission of streaming data, however, is not.

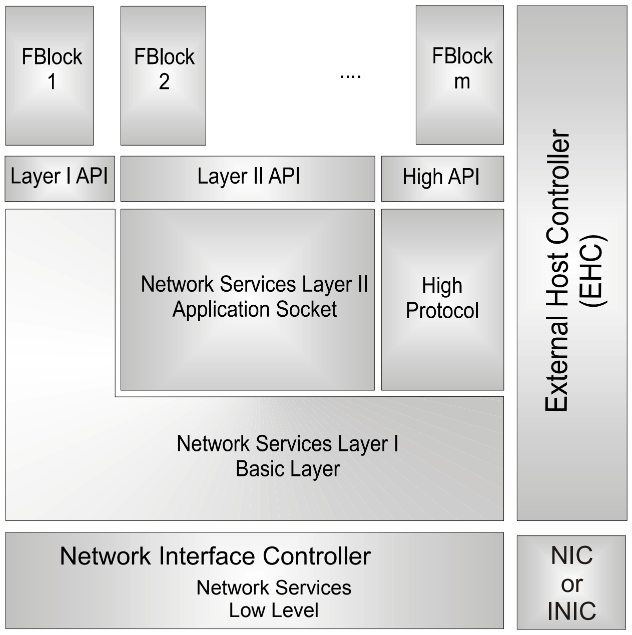

FBlock 1
FBlock 2
....
FBlock m
Layer I API
Layer II API
High API
Network Services Layer II Application Socket
High Protocol
Network Services Layer I Basic Layer
External Host Controller (EHC)
Network Interface Controller
Network Services Low Level
NIC or INIC

Fig. 9.1: Basic structure of the MOST NetServices

9.2 MOST NetServices

The *MOST NetServices* are a standard implementation of the Network Service provided by SMSC. They are tuned to the Network Interface Controllers (NIC and INIC).

The *MOST NetServices* are divided into the Basic Layer (Layer I) and the Application Socket (Layer II) and can contain additional modules, such as the *MOST High Protocol* (see section 5.8).

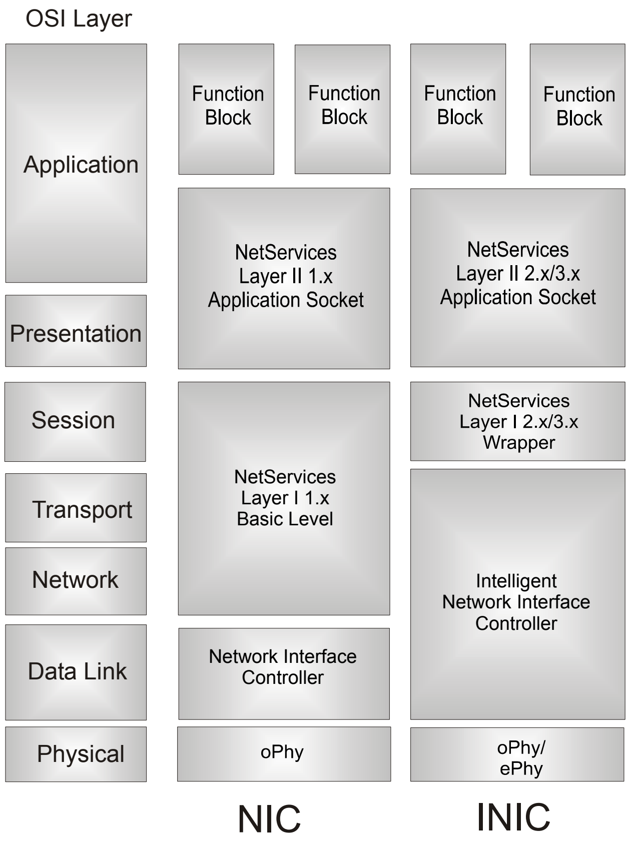

OSI Layer
Application
Presentation
Session
Transport
Network
Data Link
Physical
Function Block
Function Block
Function Block
Function Block
NetServices Layer II 1.x Application Socket
NetServices Layer II 2.x/3.x Application Socket
NetServices Layer I 2.x/3.x Wrapper
NetServices Layer I 1.x Basic Level
Intelligent Network Interface Controller
Network Interface Controller
oPhy
oPhy/ ePhy
NIC
INIC

Fig. 9.2: Model of the MOST NetServices

The MOST NetServices are written in ANSI-C and can be used directly for porting on the different processor platforms (EHC). SMSC offers 4 license types:

- Evaluation license
- System license
- Product license
- Tool license

9.2.1 Models of MOST NetServices

The NIC and the INIC have their own versions of the MOST NetServices Layer I (version 1.x for NIC; versions 2.x and 3.x for INIC), since the INIC covers the basic functions of Layer I itself and the MOST NetServices only have to provide an interface between Layer I and II. The MOST NetServices Layer II 1.x and 2.x/3.x only differ in the modules, whose functionality is covered by the INIC.

9.2.2 Dependencies on MOST Specification Baseline

The MOST NetServices versions 1.x and 2.x have been designed according to MOST Specification Rev2.x. Version 1.x is available for MOST25 only, while version 2.x is suitable for MOST25 and MOST50.

Version 3.x has been designed according to MOST Specification Rev. 3.x and is available for MOST50 and MOST150.

9.3 Modules of the MOST NetServices

9.3.1 Layer I Version 1.x

The MOST NetServices Layer I version 1.x have been designed for the NIC. The I²C bus, the SPI or a parallel interface are available as interfaces. The software interface of the NIC consists of a set of registers, which can be read and written by the microcontroller. The set of registers is referred to as a register wall.

In the following, the modules of Layer I are introduced.

Figure 9.3 gives an overview of the modules of Layer I 1.x.

MOST NetServices Kernel (MNS)

MOST NetServices Kernel (MNS) is the central entry point for all services of Layer I. It initializes all trigger and call-back functions.

MOST Supervisor (MSV)

The MOST Supervisor (MSV) provides a finite state machine for the network management. The following functions are covered:

- NIC initialization
- Determination, whether the device is TimingMaster or TimingSlave
- Configuration of the Source Port

- Network start-up and shutdown
- Power Management
- Error diagnosis and error message generation
- Self test

Different Source Port configurations are supported.

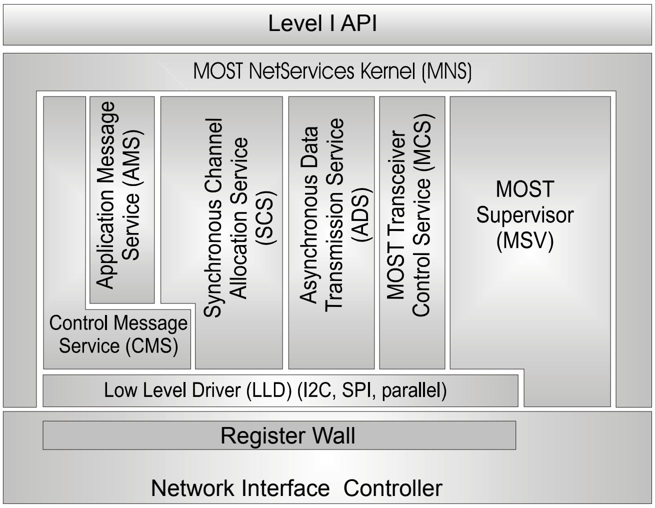

Level I API
MOST NetServices Kernel (MNS)
Application Message Service (AMS)
Synchronous Channel Allocation Service (SCS)
Asynchronous Data Transmission Service (ADS)
MOST Transceiver Control Service (MCS)
MOST Supervisor (MSV)
Control Message Service (CMS)
Low Level Driver (LLD) (I2C, SPI, parallel)
Register Wall
Network Interface Controller

Fig. 9.3: MOST NetServices Layer I version 1.x

Control Message Service (CMS)

The Control Message Service (CMS) sends and receives the control messages which are transmitted on the Control Channel. The service comprises the following functions:

- Initialization
- Control Message reception service
- Control Message sending service
- Message buffering
- Error handling and error message

Asynchronous Data Transmission Service (ADS)

The Asynchronous Data Service (ADS) is responsible for sending and receiving packet data on the Packet Data Channel. The service comprises the following functions:

- Initialization
- Buffered sending of packet data

- Buffered receiving of packet data
- Error handling and error message

Application Message Service (AMS)

The Application Message Service (AMS) is based on the Control Message Service and provides a transmission protocol for segmentation of longer application messages which are transmitted on the Control Channel. It segments the messages into 17-byte blocks in the sender and reassembles the message from these blocks in the receiver. The service provides the following functions:

- Initialization
- Sending application messages
- Receiving application messages
- Buffering messages
- Error handling and error message

Synchronous Connection Service (SCS)

The Synchronous Connection Service (SCS) controls the Routing Engine (RE) integrated into the NIC. It provides the necessary functions for allocating and releasing the Streaming Data Channels and for configurating the streaming data source and sink. The following are the functions in detail:

- Channel Allocation
- Allocation and Routing
- De-Allocation
- De-Allocation and Routing-Disconnect
- Source-Connect
- Source-Disconnect
- Sink-Connect
- Sink-Disconnect
- Detect-Channel by label

MOST Transceiver Control Service (MCS)

The Transceiver Control Service (MCS) enables the application-specific initialization of the NIC. By means of this service, hardware-relevant parameters can be set. The following functions are available:

- Setting of the address registers in the NIC
- Access to important NIC registers (via the Register Access API)

- Setting of the RMCK frequency
- TimingMaster functions (selection of the frequency source, setting of the Boundary Descriptor)

Further information can be found in the MOST Specification [MOST 3.0], in the *Basic Services (Layer I) Programmer´s Library* documentation [MNS L1 1.x] and in the *MOST NetServices Layer 1 Example* [MNS L1 Exa].

9.3.2 Layer I Versions 2.x and 3.x

The MOST *NetServices* versions 2.x and 3.x of Layer I are based on the INIC, which integrates the kernel functionality of Layer I (administration of the network status, transceiver check, driver for message reception and dispatch, control of the Routing Engine). The MOST NetServices thus only have to provide buffers for data handling and the API known from version 1.x. The modules of the NetServices are also referred to as *Wrappers*. Access to the INIC is no longer effected via a register wall, but via an integrated FBlock based on the messages. This different type of access is, however, abstracted by the wrapper modules. The Layer I API remains mostly unchanged.

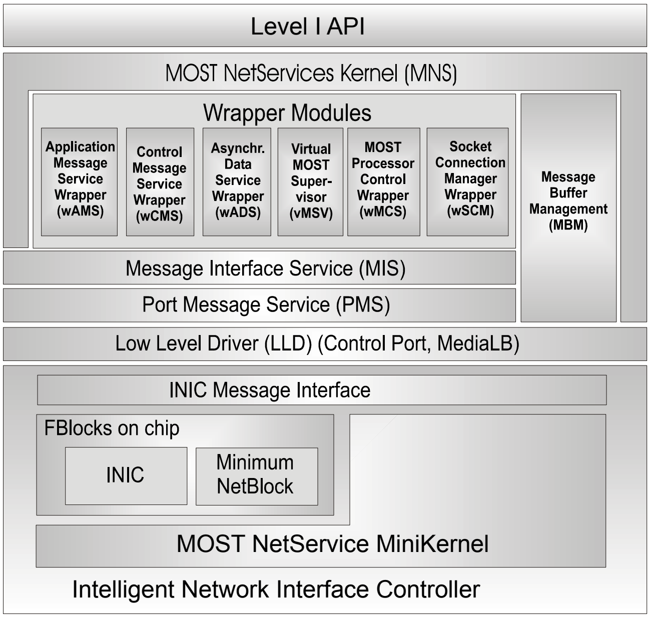

Level I API
MOST NetServices Kernel (MNS)
Wrapper Modules
Application Message Service Wrapper (wAMS)
Control Message Service Wrapper (wCMS)
Asynchr. Data Service Wrapper (wADS)
Virtual MOST Super-visor (vMSV)
MOST Processor Control Wrapper (wMCS)
Socket Connection Manager Wrapper (wSCM)
Message Buffer Management (MBM)
Message Interface Service (MIS)
Port Message Service (PMS)
Low Level Driver (LLD) (Control Port, MediaLB)
INIC Message Interface
FBlocks on chip
INIC
Minimum NetBlock
MOST NetService MiniKernel
Intelligent Network Interface Controller

Fig. 9.4: MOST NetServices Layer I version 2.x

The interface for MOST Control Messages, as well as for administrative communication between the NetServices and INIC is the Control Port, which is physically addressed via I²C, or the MediaLB Port.

A further interface is used to provide access to the Packet Data Channel. This interface can be physically addressed via I²C, MediaLB, or in version 3.x also via SPI.

Figure 9.4 shows the structure of the MOST *NetServices* version 2.x.

Figure 9.5 illustrates the structure of version 3.x. Auxiliary to the new SPI interface, the Control Message Service (wCMS) was eliminated and the MOST Debug Message Module (MDM) was newly introduced.

MOST NetServices Kernel (MNS)

MOST NetServices Kernel (MNS) is, as in version 1.x, the central entry point for all services of Layer I. It initializes all trigger and callback functions.

Virtual MOST Supervisor (vMSV)

The Virtual MOST Supervisor (vMSV) provides the user with a mapping of the actual supervisor implemented in the INIC. The functionality is similar to version 1.x. In comparison to version 2.x, version 3.x provides enhanced network diagnostics functions.

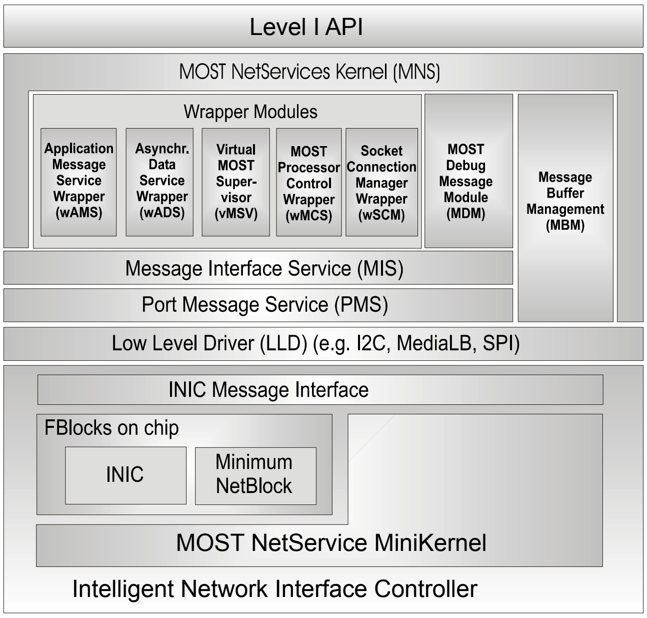

Level I API
MOST NetServices Kernel (MNS)
Wrapper Modules
Application Message Service Wrapper (wAMS)
Asynchr. Data Service Wrapper (wADS)
Virtual MOST Super-visor (vMSV)
MOST Processor Control Wrapper (wMCS)
Socket Connection Manager Wrapper (wSCM)
MOST Debug Message Module (MDM)
Message Buffer Management (MBM)
Message Interface Service (MIS)
Port Message Service (PMS)
Low Level Driver (LLD) (e.g. I2C, MediaLB, SPI)
INIC Message Interface
FBlocks on chip
INIC
Minimum NetBlock
MOST NetService MiniKernel
Intelligent Network Interface Controller

Fig. 9.5: MOST NetServices Layer I version 3.x

Wrapper Modules

The Wrapper Modules *Control Message Service Wrapper* (wCMS), *Application Message Service Wrapper* (wAMS), *MOST Processor Control Service Wrapper* (wMCS) and *Asynchronous Data Service Wrapper* (wADS) basically map the API of version 1.x onto the functions of the INIC. Only the API of the Socket Connection Manager Wrapper (wSCM) deviates considerably from the SCS-API because the administration of the Streaming Channels now runs directly via sockets and no longer via the direct programming of the Routing Engine.

The *Control Message Service Wrapper* (wCMS) applies for version 2.x only. In version 3.x it is substituted by the *Application Message Service Wrapper* (wAMS).

In comparison to version 2.x, the Socket Connection Manager Wrapper (wSCM) of version 3.x also provides interfaces and configurations to access the Isochronous Streaming Data Channel and new interfaces (e.g. TSI - Transport Stream Interface for video applications).

Data Handling to the INIC

The interface between the INIC and the *MOST NetServices* uses a Port Message Protocol. The modules *Message Interface Service* (MIS), *Message Buffer Management* (MBM) and *Port Message Service* (PMS) control this protocol, buffer the messages and supervise the different FIFOs in the INIC.

Further information can be found in the *MOST NetServices Layer I –Wrapper for INIC – User Manual/Specification* [MNS L1 2.x or MNS L1 3.x].

MOST Debug Message Module (MDM)

MOST Debug Messages are messages that are reported on the MOST Control Channel, suitable to signal any device or system related debug information. Such messages are not received by any other device, but they can be recorded by System Analysis tools. The MOST Debug Message Module (MDM) controls the protocol and provides transmission services for such events. It is available in version 3.x only.

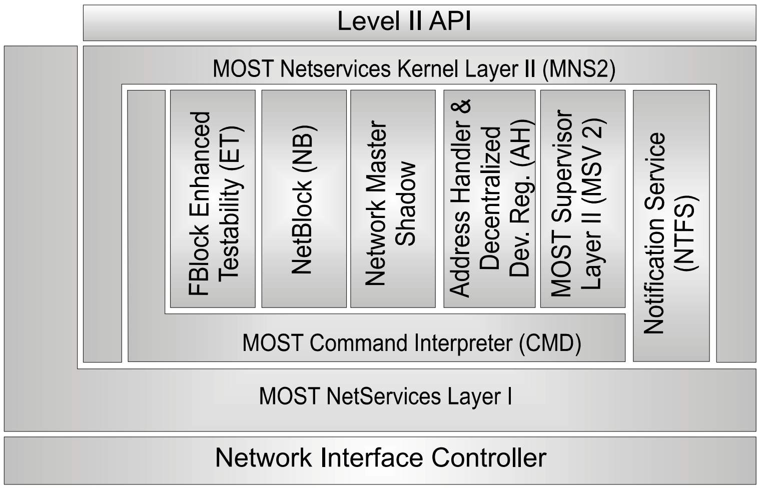

Level II API
MOST Netservices Kernel Layer II (MNS2)
FBlock Enhanced Testability (ET)
NetBlock (NB)
Network Master Shadow
Address Handler & Decentralized Dev. Reg. (AH)
MOST Supervisor Layer II (MSV 2)
Notification Service (NTFS)
MOST Command Interpreter (CMD)
MOST NetServices Layer I
Network Interface Controller

Fig. 9.6: MOST NetServices Layer II

9.3.3 Layer II Versions 1.x, 2.x and 3.x

The *MOST NetServices* Layer II builds on *MOST NetServices* Layer I. They provide the API for the application-specific function blocks. The system function blocks, such as the NetBlock, are, however, part of the *MOST NetServices*. Since the implementation of the function block *EnhancedTestability* (ET) is application-specific; the *MOST NetServices* only contain a "skeleton".

The modules which differ in version 1.x and 2.x will be referred to in the corresponding sections. Figure 9.6 shows the structure of Layer II.

MOST NetServices Kernel Layer II (MNS2)

As in Layer I, *MOST NetServices Kernel* (MNS2) is the central entry point for all services of Layer II. It initializes all trigger and callback functions.

MOST Supervisor Layer II (MSV2)

MOST Supervisor Layer II (MSV2) also provides a finite state machine for managing the network state on the application level (e.g., communication permitted/not permitted). The following functions are implemented:

- Address initialization
- Configuration on the application level after the Init Ready event
- In the NetworkSlave: Initialization of the Decentral Registry
- In the NetworkMaster: Network Change Event (NCE) supervision

FBlockID	*Pointer on InstID; stored in RAM*	*Pointer on Function Table; stored in ROM*
NetBlock	0	&Func_NetBlock
FBlock 1	&NetBlock.pFBlockIDs.InstID[0]	&Func_FBlock1
FBlock 2	&NetBlock.pFBlockIDs.InstID[1]	&Func_FBlock2
…		
FBlock n	&NetBlock.pFBlockIDs.InstID[n-1]	&Func_FBlockn
FBlock Shadow 1	&InstIDShadow[0]	&Func_FBlockShadow1
FBlock Shadow 2	&InstIDShadow[1]	&Func_FBlockShadow2
…		
FBlock Shadow m	&InstIDShadow[m-1]	&Func_FBlockShadowm

Table 9.1: Table of the available FBlocks and FBlock Shadows
n – number of FBlocks
m – number of FBlock Shadows (source: [MNS L2])

Command Interpreter (CMD)

The Command Interpreter Module is used for table-controlled decoding of received control messages and for automatically calling the handler functions of the FBlocks defined by the user.

The decoding table contains all implemented FBlocks and FBlock Shadows. It basically has the structure shown in table 9.1 and can easily be extended.

The table of the available FBlocks and the function tables can be stored in ROM. As the instance ID can be changed during runtime, the table containing these IDs must be stored in RAM (second column). During initialization, it is filled with default values. The third column contains the entry address of the related function.

Notification Service (NTFS)

The Notification Service (NTFS) is used for automatically sending status messages to other devices, when the properties of an implemented FBlock change. This mechanism was already introduced in section 4.2.2.

Table 9.2 shows the Notification Matrix of an FBlock. The matrix must be available for each FBlock supporting the Notification Service.

FBlock x				
Property 0	*Property 1*	*Property 2*		*Property (n-1)*
Flagfield 0	Flagfield 1	Flagfield 2	…	Flagfield (n-1)
Entry 0.1	Entry 1.1	Entry 2.1	…	Entry (n-1).1

FBlock x				
Property 0	*Property 1*	*Property 2*		*Property (n-1)*
Entry 0.2	Entry 1.2	Entry 2.2	…	Entry (n-1).2
Entry 0.3	Entry 1.3	Entry 2.3	…	Entry (n-1).3
Entry 0.4	Entry 1.4	Entry 2.4	…	Entry (n-1).4
…	…	…	…	
Entry 0.m	Entry 1.m	Entry 2.m	…	Entry (n-1).m

Table 9.2: Notification Matrix of the FBlock x (source: [MNS L2])
n – number of properties of the FBlock which support the Notification Service
m – max. number of devices able to subscribe to the property

The term *Entry* in the table is an index in the Device Table which stores the logical or group addresses of the devices that have subscribed to properties.

Address Handler (AH)

The Address Handler (AH) is based on a state machine. It transforms symbolic addresses into memory locations and manages the Decentral Registry in the Slave. Similar to other modules of Layer II, it closely cooperates with the Application Message Service (AMS).

The Controller does not necessarily have to know which device implements an FBlock. It suffices if the FBlockID and the InstID are known. The wildcard 0xFFFF is entered into the target address field of the control message. If the Address Handler (AH) receives a message with this wildcard, it first searches the FBlockID with the related InstID in the Decentral Registry in the standard configuration, as is shown in figure 9.7. If it finds the FBlock, it replaces the target address field with the real target address. If the search fails, it inquires, by means of the NetworkMaster Shadow Module, about the FBlock at the Central Registry located in the NetworkMaster. A further mode, where the MOST NetServices can inquire about the desired instance directly at each device in the ring is not defined in the MOST Specification and thus is neither used, nor supported by version 3.x.

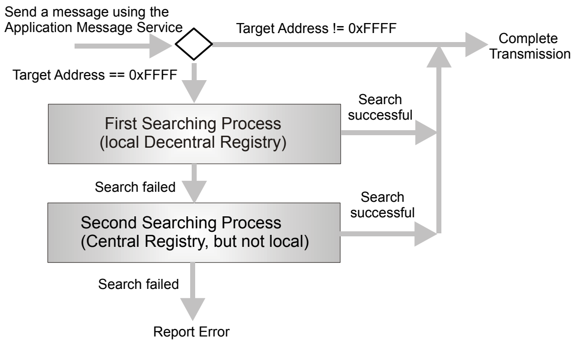

Send a message using the Application Message Service
Target Address != 0xFFFF
Complete Transmission
Target Address == 0xFFFF
Search successful
First Searching Process (local Decentral Registry)
Search failed
Search successful
Second Searching Process (Central Registry, but not local)
Search failed
Report Error

Fig. 9.7: Search procedure of the Address Handler (source: [MNS L2])

Figure 9.8 shows an example of address handling. Track 10 in the CD changer (CDC) is to be selected via the HMI. The CD changer application is represented on the device with the FBlock CD (Instance-ID: 1). To change the track, the HMI sends the control message `CD.1.Track.Set(10)`. The HMI application gives the command `0xFFFF.CD.1.Track.Set(10)` to the Address Handler, which replaces the wildcard 0xFFFF with the target address of the CD changer, according to the procedure explained above. The MOST Network Interface Controller automatically adds the source address and sends the message to the CD changer. The MOST Network Interface Controller of the CD changer sends the message on to the *MOST NetServices*. The CMD module calls the function for changing the property *Track* in the FBlock CD.

Figure 9.8 shows the general syntax of the control message on its way from the sender to the receiver.

NetworkMaster Shadow Module

The NetworkMaster Shadow Module provides a copy of the properties of the FBlock *NetworkMaster*. It reacts to status and result messages and is usually implemented on all devices, except the NetworkMaster. The basic functions are:

1. Managing the property *NetworkMaster.Configuration* during the initializing procedure.
2. Managing the property NetworkMaster.CentralRegistry during the address search procedure of the Address Handler.

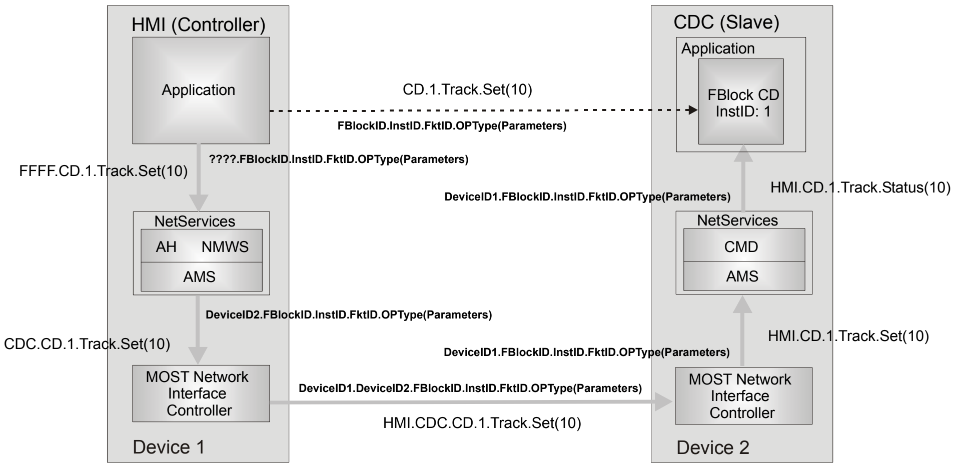

HMI (Controller)
Application
CD.1.Track.Set(10)
FBlockID.InstID.FktID.OPType(Parameters)
????.FBlockID.InstID.FktID.OPType(Parameters)
FFFF.CD.1.Track.Set(10)
NetServices
AH
NMWS
AMS
DeviceID2.FBlockID.InstID.FktID.OPType(Parameters)
CDC.CD.1.Track.Set(10)
MOST Network Interface Controller
Device 1
DeviceID1.DeviceID2.FBlockID.InstID.FktID.OPType(Parameters)
HMI.CDC.CD.1.Track.Set(10)
CDC (Slave)
Application
FBlock CD InstID: 1
DeviceID1.FBlockID.InstID.FktID.OPType(Parameters)
HMI.CD.1.Track.Status(10)
NetServices
CMD
AMS
DeviceID1.FBlockID.InstID.FktID.OPType(Parameters)
HMI.CD.1.Track.Set(10)
MOST Network Interface Controller
Device 2

Fig. 9.8: Example of address handling (according to [MOST 2.5] and [MOST 3.0])

NetBlock (NB)

The NetBlock (NB) is a system FBlock, which must be implemented on all devices. It is thus part of the *MOST NetServices* Layer II. It was already introduced in section 4.3.4.

Due to the fact that a minimum NetBlock (NBMIN) is already integrated in the INIC, only the *NetBlock EHC* (NBEHC) of the NetBlock in Layer II version 2.x or 3.x is implemented (see section 10.4.2).

FBlock EnhancedTestability (ET)

The FBlock EnhancedTestability (ET) is a system FBlock as well and is also described in section 4.3.4.

9.4 Integration of the MOST NetServices onto the Target Platform

The MOST NetServices were written in ANSI-C without using any standard libraries and can thus very easily be ported onto the target platform. The configuration and adaptation for the hardware and software platform used is effected via compiler switches in the header files *adjust*.h*. This enables a good scaling onto the processor architecture. The *MOST NetServices* can be implemented on an 8-bit microcontroller with only a few kbytes RAM in a Main Loop Architecture, or also on a 32-bit microcontroller with adequate multitasking software systems, such as QNX, WinCE, or Linux.

The Low Level Driver (LLD) shown in Figures 9.3 and 9.4 must be implemented by the user. The LLD is used for abstraction of the hardware with regard to the *MOST NetServices*. The two System-On-Chip Controllers OS8805 and OS88558 are an exception. In this case, the MOST NIC and the Host Controller are integrated on one chip. The *MOST NetServices* only need to be configured by the user.

9.4.1 Callback Functions

The largest part of the *MOST NetServices* provides callback functions for status and error messages. These functions are made available to the programmer and are called by the *MOST NetServices* if a specific event or an error has occurred.

The callback functions are characterized in the *MOST NetServices* documentation by the key word [*Callback*].

```
[Callback]
declaration of function;
```

The *MOST NetServices* offer different types of callback functions for the various versions.

MOST NetServices Layer I 1.x

The *MOST NetServices* Layer I 1.x specify callback functions of type A and B:

- Even though the type A functions must be defined by the programmer, the function body can remain empty if the event which calls the function is not relevant for the application.
- Type B functions, however, must always be implemented completely, since they are responsible for system-relevant events, such as *PowerOn*.

MOST NetServices Layer I 2.x and 3.x

In the case of the *MOST NetServices* Layer I 2.x and 3.x, the callback functions for unused events do not have to be declared. The *MOST NetServices* ensure a defined behavior. It is differentiated between *module specific standard callback* and *general standard callback functions*:

- The module specific standard callback functions are called for a specific event by only one module.
- The general standard callback functions, however, can be called by several modules. They usually indicate the result of interactions with the network interface.

MOST NetServices Layer II 1.x, 2.x and 3.x

Like the *MOST NetServices* Layer I, Layer II also defines type A and type B functions. Additionally, version 1.x and 2.x define type C functions. Type C functions must be explicitly activated by the programmer in a header file (adjust*.h).

9.4.2 Main Loop Architecture

On small processors without a multitasking system software, all applications and services – except for Interrupt Service Routines – are usually called cyclically from a loop in the main function of the software. The programmer must ensure that for all applications and services the maximum response time until the next call of their central entry function is observed and that the maximum execution time until the next return to the Main Loop is not exceeded.

For *MOST NetServices* 1.x, the cyclical call of the services must be within 15 ms. Otherwise, central functions of the MSV, such as checking the lock status and similar operations, can no longer be assured to meet the MOST Specification. The maximum return time is typically less than 1 ms, unless the *MOST NetServices* call callback functions of the applications on their part. In that case the application programmer is responsible for ensuring a return into the *MOST NetServices*, and thus into the Main Loop, within the required time.

For the *MOST NetServices* 2.x and 3.x, the problems are easier to handle because all time-critical functions are taken over by the INIC itself and the *MOST NetServices* running on the host basically only form the interface with the application software. The maximum response time also has to be observed in this case, particularly because the INIC supervises the communication with the host by means of a Watchdog Timer which is typically configured to half a second. If the INIC does not receive a response from the host within this time, it assumes an error and disconnects the host from the network. This results in all applications (FBlocks) provided by the device in the network being unregistered.

In addition to the main entry functions of the *MOST NetServices* in the Main Loop, other functions have to be called which provide the MOST NetServices with information as to the timing. They are usually triggered by a timer-controlled Interrupt Service Routine (MOSTTimerInt). In version 1.x, call frequencies of 1, 10 and 25 ms can be selected. The *MOST NetServices* 2.x and 3.x have an advanced timer implementation that supports more flexible call frequencies. which uses the timing information provided by the INIC.

9.4.3 Multitasking Architectures

The items mentioned above can also be applied to multitasking architectures. The Main Loop is, however, replaced by a Task Loop. Usually, the *MOST NetServices* run in a loop which, apart from the entry functions mentioned above, only calls functions necessary for communication with other tasks or drivers. Functions for serving other services and applications are typically integrated in additional tasks.

For the *MOST NetServices* task, the priority must be selected such that the minimum call frequency of the *MOST NetServices* 1.x can be guaranteed. Higher priorities will generally only be reserved for tasks with a real-time functionality (such as audio/video signal processing, timer) or for network drivers with high timing requirements (such as CAN network driver stack). The actual application should normally run in tasks with a lower priority.

Due to the fact that the MOST NetServices are written in ANSI-C, they are not thread-safe. This means that the API functions may only be called from *one* context. This means that the handler functions of the CMD module must be called and the data must be accessed in the context of the MOST NetServices tasks and not in the context of the respective application. The programmer must solve this problem by using suitable synchronization mechanisms of the system software, such as semaphores. For this purpose, the *MOST NetServices* 2.x and 3.x offer macros in the configuration files which can be assigned to platform-specific Mutex mechanisms (Mutual Exclusion).

For larger systems with several applications, one could imagine using, instead of the central command interpreter, an architecture which passes the messages received from the AMS and the *MOST High Protocol* through task borders. One could also imagine executing the evaluation of the messages (also a function of the CMD module) in the context of the respective application task. Such architectures are also available by third-party providers.

9.4.4 System Management Module

Modules such as the NetworkMaster, PowerMaster and ConnectionMaster are not part of the *MOST NetServices*. They can be licensed as reference implementations from SMSC in the same manner as the *MOST NetServices*. In contrast to the *MOST NetServices*, however, implementations of the device manufacturers are also commonly used here.

10 MOST Interface Controller

10.1 MOST Network Interface Controller and MOST Device

Within a MOST device, the MOST Interface Controller possesses the role of a two-way bridge between the MOST network and the interior of the device. Figure 10.1 shows an example for the structure of a typical single-processor style MOST device. The application hardware provides the basis for the characteristic features of the MOST device. In this case, a Digital-to-Analog Converter (DAC) followed by an amplifier stage implies the MOST device to be an amplifier.

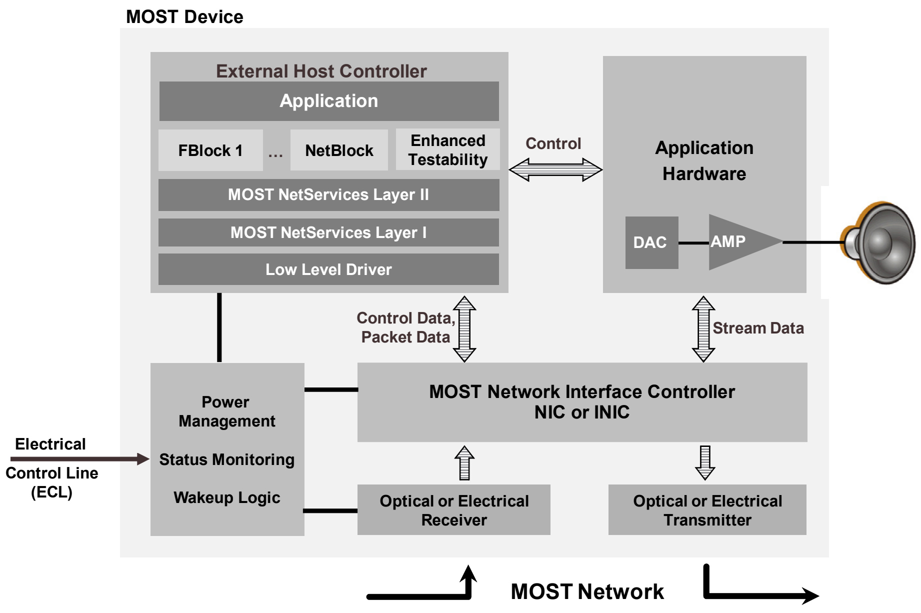


Fig. 10.1: Structure of a MOST Device

An External Host Controller (EHC) processes all software tasks, including controlling of the application hardware and running MOST related routines.

Towards the MOST network, there is a sending path and a receiving path. In the latter, a receiver translates the signals coming from the MOST network into a form that can be processed by the MOST Interface Controller. In the sending path, the output of the MOST Interface Controller is converted. This is achieved by an external circuitry that generates a signal compliant to the underlying physical layer specification, depending on the chosen physical layer. Signal transmission then happens either in the optical domain (optical physical layer), or in the electrical domain (electrical physical layer).

In order to allow the MOST device to be woken by activity on the network, the receiver is connected to the wake-up area.

The MOST Interface Controller provides the required standardized interfaces for connecting to the application as well as the MOST network. This allows to send and receive streaming, control, packet and isochronous data via the MOST network in an efficient way. On the MOST side, the MOST Interface Controller receives the bit stream from the physical layer, refreshes this signal, modifies the data if required and emits the signal through the sending path. The refreshing is done to compensate for signal degradation on the physical layer, in order to re-establish signal integrity for achieving best transmission results.

An important part of the signal handling on the MOST side is synchronization. The MOST Interface Controller either creates the system wide clock (when being configured as the sole TimingMaster of the system), or synchronizes to the incoming bit stream (common for the majority of MOST devices). Thus, all MOST devices in a MOST network are synchronized. This allows for an efficient high quality signal transmission running at a minimum of overhead.

Device wake-up, power management and status monitoring make sure that the MOST device complies with the MOST specification in terms of start-up, power consumption in sleep mode and handling of exceptional situations in power supply.

10.2 MOST Network Interface Controller Families

There are two types of MOST Interface Controllers: the Intelligent Network Interface Controller (INIC) and the Network Interface Controller (NIC). While the NICs represent the first implementations of a Network Interface Controller, their role in development of new MOST systems and devices meanwhile has been assumed by the INICs for several reasons. One of the most important motivations for this trend is the increase of robustness and simplification on system level that is associated with the usage of the INICs.

10.2.1 INIC

INICs act as simple MOST devices with regard to the MOST network and its management. They can act autarchically, i.e., independent of the EHC, in order to speed up system start and stabilize the MOST system at the earliest possible point in time.

This is a great advantage, particularly if the application on the EHC has a long boot time, or if an error occurs in the application during normal operation.

For the INIC, the function of the entire network has the highest priority. It enables the network to start up in a controlled manner and it protects the network from undesired side effects in the case of errors in the application software (reset or "hang-up"). The name of the related operational mode – Protected Mode – refers to this fact. This status is adopted accordingly after a hardware reset of the INIC and during the Power-Up.

The INIC is responsible for the basic performance of the MOST device in which it is utilized. The EHC and the application running on it boot up independent of the INIC start-up and log on to the INIC upon completion of their initialization.

There is a common API available for controlling the INIC. Communication between the EHC and the INIC is based upon the Port Message Protocol. The specific functions of the INIC are modeled as a function block, in accordance with the general approach to MOST devices. The routing of streaming data can easily be controlled via sockets and connections. The application itself does not need to use the INIC-API directly, but is based on the MOST API (MOST NetServices). The MOST NetServices control the INIC via the INIC-API using the Port Message Protocol. According to the MOST philosophy, control of the INIC is organized by means of a function block (FBlock) called FBlock INIC.

INICs have a fast serial and synchronous interface for streaming, control and packet data, which is referred to as MediaLB. This interface can be used, for example, to connect several audio components on a printed circuit board, providing them, e.g., with streaming data. The INIC drives the basic clocking of the MediaLB synchronized to the MOST network. This interface is implemented in all INICs.

Additional, standardized interfaces allow connecting application hardware and the INIC, e.g., through I^2S.

The properties described in this section make the INIC a first choice component as far as the implementation of new MOST devices and MOST based multimedia systems are concerned.

10.2.2 NIC

Network Interface Controllers (NICs) were the first generation of communication ICs for MOST. They have meanwhile been supplemented by the INICs, which enable a more robust system design and a faster product realization. Within the MOST devices, NICs cover the data link layer functionality. They are controlled via a Register Wall (register structure) by means of read/write access to individual storage positions.

In cooperation with a NIC, the EHC is fully responsible for the basic behavior of the MOST device in which it is implemented. During implementation, it must be taken care of that boot times of the application are kept short enough to ensure compliance with the MOST specifications. If there are errors in the application domain, they can, in the case of NIC-based devices, have an effect on the entire system. This was taken into consideration when developing the INIC. As a matter of principle, the INIC encapsulates such application errors locally.

The routing of streaming data is controlled via the registers of the MOST Routing Table (MRT). A typical NIC is the OS8104A (MOST25).

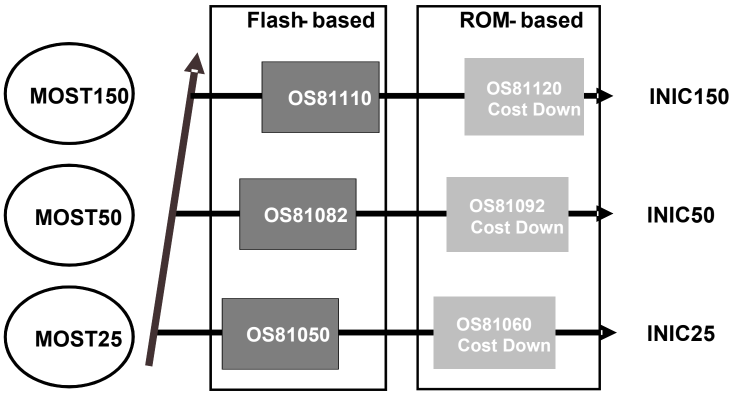


Fig. 10.2: INIC Family Overview and Roadmap

10.3 The INIC Family

10.3.1 Members

For each of the three MOST speed grades - MOST25, MOST50 and MOST150 - two INICs are available (see fig. 10.2).

There are flash memory based versions, and ROM based versions that are cost reduced variants of the flash memory based versions. In the flash memory based versions, both, power-up configuration (Configuration String) and the entire firmware of the INIC can be re-written. The ROM based versions keep their firmware in ROM and configuration information in OTP memory. The two-part structure of the OTP memory area of OS81092 allows writing the Configuration String twice.

MOST150 constitutes an exception to the description above, since only the flash based version OS81110 is currently available. The cost reduced version OS81120 is at the planning stage.

10.3.2 Overview of Features

Table 10.1 outlines the general features available in the members of the INIC family. Note that not all interfaces listed in Table 10.1 are available at the same time. Availability depends on configuration, since some of the interfaces share pins to keep pin count low.

Feature	*Data types and Interfaces*	*INIC25 OS81050*	*INIC25 OS81060*	*INIC50 OS81082*	*INIC50 OS81092*	*INIC150 OS81110*
Supported MOST Data Types	Control, Synchronous, legacy async. packets	X	X	X	X	X
	Isochronous, Ethernet					X
Interfaces (See text)	I²C, I²S, MediaLB 3-pin	X	X	X	X	X
	MediaLB 5-pin (legacy)	X	X	X	X	
	MediaLB 6-pin, TSI, SPI					X
Power Supply		3.3V / 2.5V	3.3V / 1.8V	3.3V / 2.5V	3.3V / 1.8V	3.3V / 1.8V
Package		ETQFP 44 QFP 44	QFN 40	ETQFP 64	QFN 48	QFN 48

Table 10.1: Features of the INIC Family

10.4 Interfaces to the INIC

10.4.1 Available Interfaces

Regardless of the MOST network speed grade, all members of the INIC family have a similar general structure. The INIC processor - shown in the center of Figure 10.3 - transmits the data from and to the MOST network and takes care of the routing of the data from and to the peripheral interfaces.

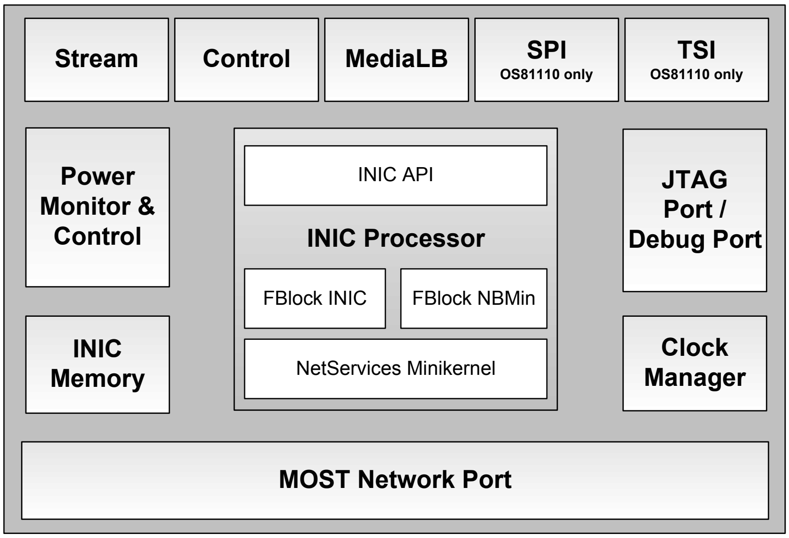


Fig. 10.3: INIC Block Diagram

The clock manager is responsible for providing the clock to the entire chip, the application hardware and – once the INIC is configured in TimingMaster mode – also for the entire network. It ensures overall synchronization.

The Power Monitor & Control module establishes the connection to an external Power Management circuit. The INIC can thus react to changes in the power supply in accordance with the MOST Specification.

By means of the JTAG / Debug Port, the INIC can be included in boundary scan testing. The same interface supports debug level access to the INIC. Connected to an INIC Explorer, internal data like power-up configuration and property values, stream data routing information and interface configurations can be retrieved. The possibility to write the Configuration String on flash based and ROM based INICs is very useful too. It makes sense to implement at least the footprint for the JTAG/Debug Port related connector (not populating it in mass production) on the printed circuit board (PCB).

The INIC Memory block shown in Figure 10.3 actually stands for different kinds of memory. It keeps continuously changing data, as well as start-up configuration data (Configuration String) and special data that needs to be kept between the system's life cycles during sleep phases.

External devices like analog-to-digital converters (ADCs), or digital-to-analog-converters (DACs), or DSPs, or similar, interface to the Streaming Port. The Streaming Port can source or sink streaming data in various industry-standard formats.

All INICs have a MediaLB interface for streaming, control, packet and isochronous data. This interface can be used to connect different audio or video components on a printed circuit board, using the INIC as a bridge, to the MOST domain. Since the clocking on MediaLB is synchronized to MOST, the synchronous network approach of MOST is maintained even at PCB level. The kinds of data available on MediaLB and the available MediaLB interface types (MediaLB 3-pin, MediaLB 5-pin, MediaLB 6-pin) depend on the used INIC and its configuration.

Industry devices like satellite tuner devices, receivers, displays, or video decoders, typically handle Transport Stream Interface (TSI) data streams. OS81110, the INIC for MOST150, has a special interface to establish a connection with such devices for transporting TSI streams over MOST, the TSI port.

Many microcontrollers and peripheral devices make use of the serial peripheral interface (SPI). The SPI port of the OS81110 is implemented as the typical four pin interface. It uses an additional interrupt pin for flow control and can transport either asynchronous packet data, or QoS IP packets (isochronous).

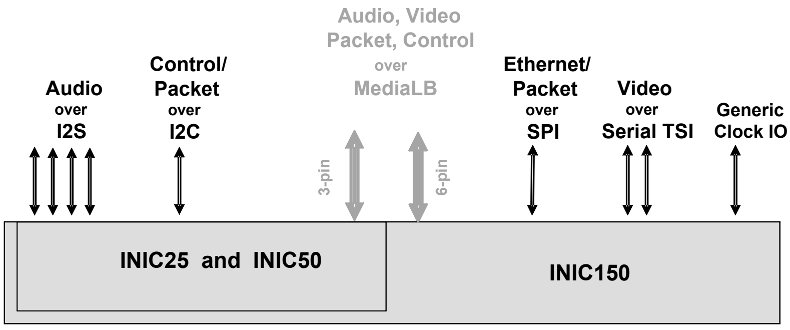


Fig. 10.4: INIC Connectivity Overview Diagram

The I²C based Control Port allows direct communication between the EHC and the INIC. It transports data based on a message based protocol. Through the Control Port, the EHC can access FBlock INIC and can send and receive legacy packet data and MOST control messages (MCM).

Last but not least, all INICs establish the connection to the MOST network at the MOST speed grade they are designed for. The MOST Network Port is specifically designed to accomplish this task and therefore represents the interface to the particular MOST network. It recovers the MOST network clock, decodes incoming data, encodes data in the outgoing direction and refreshes the signals towards its output to achieve the best possible transmission quality.

10.4.2 Interfaces from Software Perspective

Overall Structure

All internal functions of the INIC are accessible through the so-called INIC API, using the Port Message Protocol. The specific internal functions of the INIC are modeled as a function block (FBlock INIC), in accordance with the general approach to MOST devices. Function block NetBlockMin (NBMIN) allows the INIC to act autarchic towards the Network on absence of a running application software. The MOST NetServices MiniKernel contains crucial MOST NetServices functionality that relieves the EHC of real-time style supervision tasks. It also contributes to the capability of the INIC to act independent of the EHC. The application itself does not use the INIC-API directly, but is based on the MOST API (MOST NetServices). The MOST NetServices control the INIC via the INIC-API and Port Message Protocol.

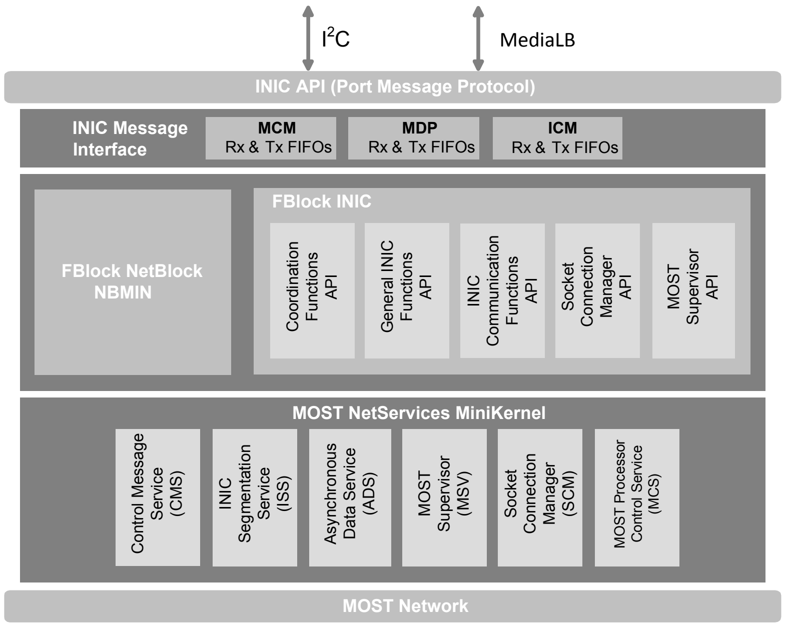

I²C
MediaLB
INIC API (Port Message Protocol)
INIC Message Interface
MCM Rx & Tx FIFOs
MDP Rx & Tx FIFOs
ICM Rx & Tx FIFOs
FBlock NetBlock NBMIN
FBlock INIC
Coordination Functions API
General INIC Functions API
INIC Communication Functions API
Socket Connection Manager API
MOST Supervisor API
MOST NetServices MiniKernel
Control Message Service (CMS)
INIC Segmentation Service (ISS)
Asynchronous Data Service (ADS)
MOST Supervisor (MSV)
Socket Connection Manager (SCM)
MOST Processor Control Service (MCS)
MOST Network

Fig. 10.5: INIC Software Stack Overview

INIC Message Interface

For communication towards the EHC, the internal structure of the INIC is organized in a threefold way. The INIC Message Interface consists of three sets of buffers that correspond to three different kinds of messages.

MOST Control Messages (MCM) are identical to MOST Control Messages that travel over the MOST network for the purpose of control level communication.

MOST Data Packets (MDP) and MOST Ethernet Packets (MEP) both refer to packetized data. While MDP belongs to the legacy packet data that is available on MOST networks since the implementation of MOST25, MEP contain packet data for Ethernet style data transfer.

INIC Control Messages (ICM) have the same structure as MCM, but are directed only towards FBlock INIC. The EHC uses ICM to control and configure the INIC during runtime, while the INIC replies accordingly, e.g., with status reports.

Function block NBMIN and FBlock INIC

As explained in the introduction to this chapter, INICs can act as simple MOST devices with regard to the MOST network and its management. Since every MOST device must stringently contain the functionality of the function block (FBlock) NetBlock, the minimum version of a NetBlock, the NetBlockMin (NBMIN) is integrated into the INIC, too. Together with the NetBlock EHC (NBEHC) of the application it forms the normal NetBlock (see fig. 10.6).

NBMIN has a proxy like role, when the EHC, the MOST NetServices and the application are present. If the INIC needs to act independent of the EHC, it takes full responsibility and makes the MOST device a device without function blocks.

The FBlock INIC is the lean interface to control INIC hardware functionalities and the MOST NetServices MiniKernel. It allows access, e.g., to the Notification function of the INIC, to information such as version info and to the control mechanisms for streaming data.

MOST NetServices MiniKernel

The routines of the MOST NetServices MiniKernel handle, e.g., MOST control Messages and data packets. They configure the INIC on start-up and control, e.g., the routing of stream data. The MOST Supervisor has an important role in the MOST NetServices MiniKernel. It handles dynamic network events, network supervision, power management and errors. In general it controls the behavior of the INIC in the MOST network context and therefore takes care of, e.g., start-up and shutdown, configuration of the INIC as TimingMaster or TimingSlave.

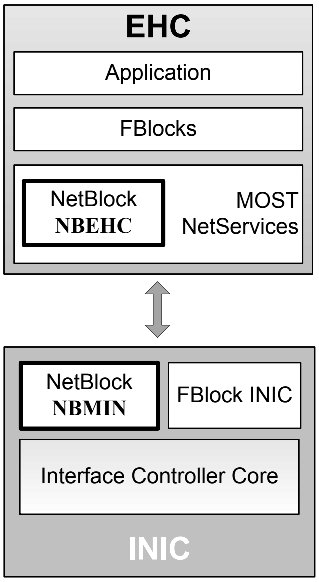


Fig. 10.6: NetBlock NBMIN and NetBlock NBEHC

10.4.3 Ports, Sockets and Connections

From the perspective of the INIC, the periphery and the MOST network are connected via ports. Their start-up configuration is partially achieved via hardware pins, but mainly by access to the properties and methods of the FBlock INIC via the INIC-API. While the Network Port – the gate to the MOST world – can transport all MOST data types available, other ports are specialized to one or more data types. Depending on the configuration, the hardware of the ports is able to move several data streams in different directions.

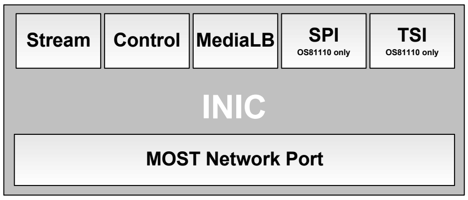


Fig. 10.7: Ports of the INIC

Ports need to be opened. Some ports are opened automatically, based on the configuration of the INIC. Other ports are opened by accessing the OpenPort function of the INIC API.

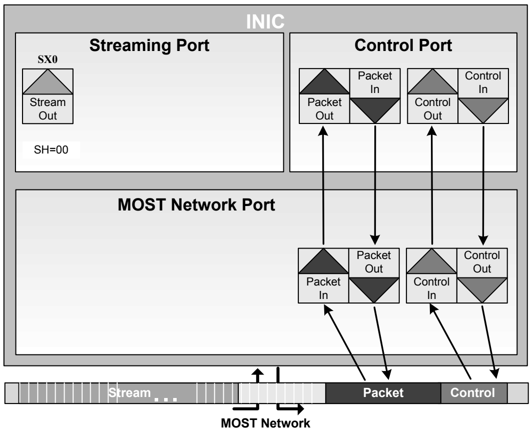


Fig. 10.8: Sockets at the MOST Network Port (MOST25). SH – Socket-Handle

Which data type is to be accessed directly and which direction is desired (IN or OUT) is defined by means of sockets.

Note: Data directions are always described from the perspective of the INIC. If data move into the component, this corresponds to the direction IN, if they flow out of the component, this corresponds to the direction OUT.

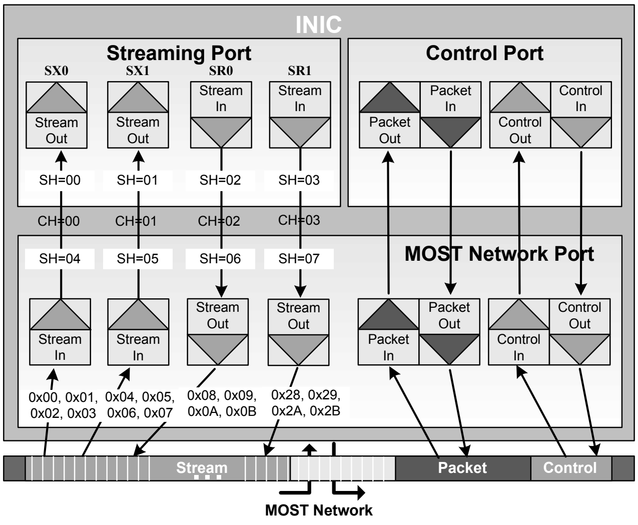


Fig. 10.9: Ports, Sockets and Connections

If a socket is generated, the INIC returns a confirmation message and an identification (Socket Handle [SH]). Establishing a connection then enables the data flow. The function for establishing the connection requires the Socket Handle of the

IN-Socket and the one of the OUT-Socket only. Establishing the connection results in a Connection Handle, which is required for connection management (also for removing the connection).

Note: It is only possible to connect sockets of type IN and OUT that have the same type of data and the same parameter settings.

10.4.4 Port Message Protocol

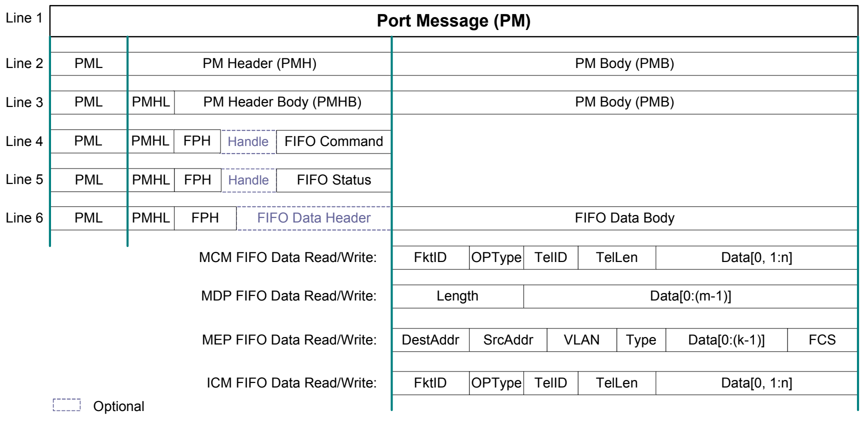


Fig. 10.10: Port Message Protocol Structure

Message exchange between INIC and the EHC is based upon the Port Message Protocol. Figure 10.10 depicts the elements of the Port Message Protocol, some of which are optional. A Port Message (PM) (see “Line 1” in fig. 10.10) consists of several parts.

The Port Message Length (PML) (see “Line 2” in fig. 10.10) is a 16-bit value, which counts the bytes of the PM Header (PMH) and the PM Body.

The PMH (see “Line 3” in fig. 10.10) consists of the eight-bit PM Header Length (PMHL) value and the PM Header Body (PMHB). PMHL counts the bytes of PMHB. PMHB either contains FIFO commands (see “Line 4” in fig. 10.10), FIFO status information (see “Line 5” in fig. 10.10), or information on the subsequent data (see “Line 6” and below in fig. 10.10).

The FIFO Protocol Header has an important function in differentiating between the different message types that are available:

- FIFO Command messages
  Used to resynchronize INIC’s interface or retry failed messages

- FIFO Status messages
  Used for handshaking and to indicate that INIC's initialization is complete
- MOST Control Messages (MCM)
- MOST Data Packets (MDP)
- MOST Ethernet Packets (MEP) OS81110 only
- INIC Control Messages (ICM)

If the PM Body is available in a particular PM, it contains the message data itself, according to the specific structures.

10.5 Protection, Encapsulation and the EHC

10.5.1 Interaction between INIC and the EHC

In order to protect the network from undesired side effects, the cooperation between INIC and EHC is controlled via a state machine, the External Host Controller Interface State Machine (EHCI-State).

The EHCI State Machine can reach three states (see fig. 10.11):

- EHCI-Protected
- EHCI-Semi-Protected
- EHCI-Attached

EHCI-Protected

The INIC takes on the EHCI-Protected basic state after a reset. It effectively prevents a starting or incorrectly running application from accidentally having access to the MOST network. The application should finally reach the normal operating mode, state EHCI-Attached, via the EHCI-Semi-Protected state.

In state EHCI-Protected it is not possible for the EHC to send messages to other nodes in the MOST network. The EHC cannot receive messages from the MOST bus either. The safety concept of the INIC provides for the application being watched by the INIC. For this purpose, the INIC has a so-called Watchdog mechanism. The INIC-API allows the configuration of the Watchdog and the change of state of the EHCI State Machine while in the EHCI-Protected state.

A handshake procedure makes sure that both, INIC and EHC, are informed about the readiness of the respective partner. If they are ready, the application running on the EHC configures the Watchdog mechanism (if required) and starts changing to the next state.

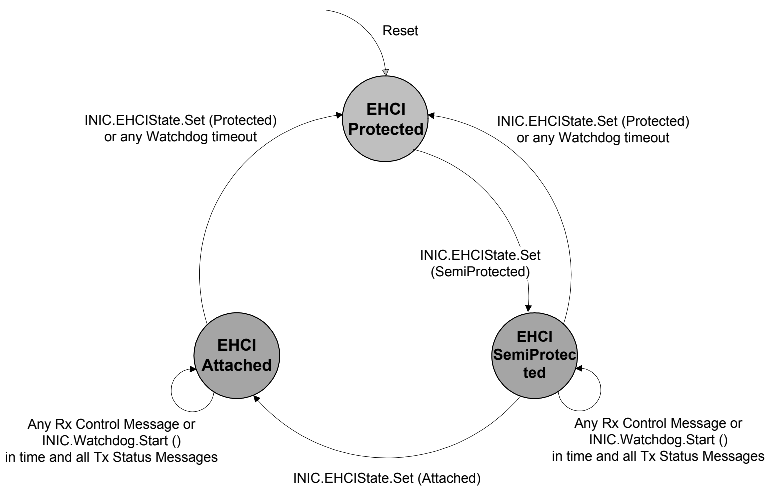

Reset
EHCI Protected
INIC.EHCIState.Set (Protected) or any Watchdog timeout
INIC.EHCIState.Set (Protected) or any Watchdog timeout
INIC.EHCIState.Set (SemiProtected)
EHCI Attached
EHCI SemiProtected
Any Rx Control Message or INIC.Watchdog.Start () in time and all Tx Status Messages
Any Rx Control Message or INIC.Watchdog.Start () in time and all Tx Status Messages
INIC.EHCIState.Set (Attached)

Fig. 10.11: The states of the EHCI State Machine

EHCI-Semi-Protected

The state EHCI-Semi-Protected also prevents communication with the MOST network. It allows, however, full access to the FBlock INIC. The underlying concept takes into consideration that the full initialization of the hardware of a MOST device also requires the start-up of such applicative parts of the connection, which possibly require, e.g., a continuous clocking.

In this state, the EHC can therefore configure and open ports, such as the Streaming Port. When opening the ports, the respective hardware interfaces start operating, thus also providing the clocks for external synchronizing purposes.

If all hardware initializations are finished, the change to state EHCI-Attached can be performed.

EHCI-Attached

In the state EHCI-Attached, the EHC has full, bi-directional access to all data types of the MOST network. The application is also fully available for the network and the applicative function blocks (FBlocks) must be registered at the NetworkMaster.

Consequences of EHCI State and NetBlock

In the state EHCI-Protected and the state EHCI-Semi-Protected NBMIN takes on the role of the main contact with regard to administrative queries from the network. Due to the fact that certain information is only available when the application is ready for operation, certain queries are acknowledged either with an empty list or with an error message.

The NBMIN also answers corresponding administrative queries in the state EHCI-Attached. Application-related queries to the EHC and its NetBlock (NBEHC), however, are then passed on to the application.

10.5.2 The Watchdog Functionality

In general, watchdog circuits are used to supervise applications with regard to malfunctions. For that purpose, a signal is implemented in the application, which, in normal operation, appears cyclically. This signal is applied to the watchdog circuit. If the signal fails to appear, this can be interpreted as an indication of a malfunction. For eliminating a malfunction, discrete watchdog circuits often trigger a hardware reset and consequently a re-initialization of the complete application.

In the case of the watchdog mechanism implemented in the INIC, messages which are exchanged between INIC and application, take on the role of the signal. Supervision is implemented individually for the sending and receiving direction. The time interval, which is set by the application during initialization and used during runtime, is valid for both directions.

When data is received from the EHC, the INIC continuously supervises whether the application regularly sends valid messages within the set time interval. Usually, the application communicating to the MOST network during normal operation ensures the necessary signaling. Should communication with the network not be necessary, the application will continue to trigger the watchdog.

In the sending direction, supervision only takes place if the INIC has sent a message to the application. In this case, the INIC supervises that the acknowledgement of reception for the sent message arrives within the defined time interval.

If the INIC detects a violation of the time interval, it protects the network from effects resulting from the obvious malfunction of the application by the EHCI State Machine carrying out a change to the EHCI-Protected state. In addition, the INIC has a dedicated reset output for the application. When this option is activated, by a respective configuration by the application, the INIC then triggers a hardware reset, e.g., of the EHC.

10.6 Configuration through Configuration String

Since the completion of the EHC start-up usually takes some time, there is a Configuration String inside the INIC, which contains a sequence of values that allow controlling the initialization of hardware and firmware of the INIC at the earliest possible moment during MOST device start-up.

Accessible Parameter	Factory Default
NBMIN.NodeAddress.NodeAddress	0xFFFF
NBMIN.GroupAddress.GroupAddress	0x03C8
NBMIN.PermissionToWake.WakeStatus	Off
NBMIN.RetryParameters.RetryTime	11
NBMIN.RetryParameters.RetryNumbers	6
INIC.VersionInfo.ConfString (Major)	0
INIC.VersionInfo.ConfString (Minor)	0
INIC.VersionInfo.ConfString (Release)	0
INIC.RMCK.Divider	Disabled
INIC.RemoteAccess.AccessMode	False
INIC.WatchdogMode.AutoShutDownDelay	65535
INIC.AddressConfig.Mode	0x00
INIC.PortConfiguration.MediaLBClockCfg	512Fs
INIC.PortConfiguration.I^{2}CSlaveAddress	0x40
INIC.PortConfiguration.DefPort	INT_Select
INIC.PortConfiguration.PortVariantCfg	Configuration 7
INIC.PortConfiguration.SerialInterfaceCfg	InOut
INIC.SCMConfig.SCMCfg	0x00
INIC.Bandwidth.AssignBWInit	80
INIC.DeviceMode.DeviceMode	Slave
INIC.RBDOptions.Options	0x01
INIC.NIWakeUpMode.Mode	0x00
INIC.PMIConfig.Config	0x00
INIC.PMIConfig.TimePwrOff	5 s
INIC.UseStatusPin	True

Fig. 10.12: Exemplary INIC Configuration String of OS81110

The Configuration String can be modified either through the ECH, or through INIC Explorer. On ROM based circuits, the Configuration String can be written twice. Flash based INICs allow modifying the Configuration String as required.

10.7 Control Port

The Control Port is based on the I²C-Bus, i.e., on serial data transmission by means of a clock line, a data line and an interrupt line. With regard to the I²C, the INIC behaves as I²C-Slave supporting Clock Stretching.

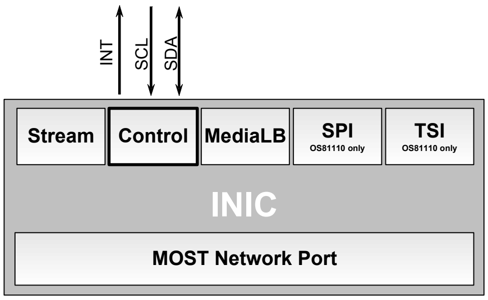


Fig. 10.13: INIC Control Port

The INIC signals the availability of data by means of an Interrupt Pin (INT). The Control Port transports control and/or packet data between INIC and EHC. In addition, it can be used to write firmware and configuration data to the INIC. The I²C address used for the Control Port is specified in the Configuration String.

10.8 Streaming Port

The Streaming port connects INIC to multimedia sources and/or sinks using serial point-to-point connections. It is synchronized to the MOST network and provides up to 4 serial data pins, which are configurable with respect to direction, data rate and data format. The Streaming port is configured through INIC API.

It supports several different data formats and provides 2 clock pins to synchronize external hardware to the MOST clock. Maximum clock speed is 512 Fs (for OS81110 only).

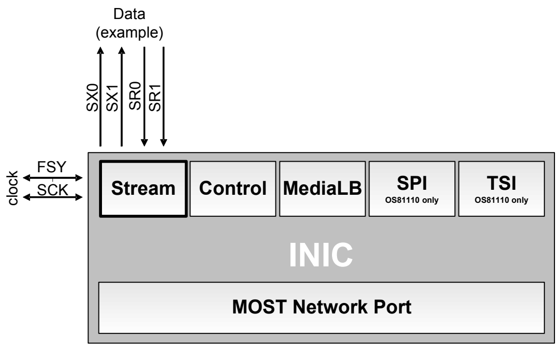


Fig. 10.14: INIC Streaming Port

All data signals of the Streaming Port share one clock pin (SCK) and one synchronization pin (FSY; for differentiating between Left/Right). If FSY and SCK are configured as output, they drive the serial data transmission on the Streaming Port. In this case, the externally connected peripheral components have to synchronize to FSY and SCK. It is possible to configure FSY and SCK as inputs. When doing so, the synchronization of these signals must take place via the RMCK (Recovered Master Clock) output of the INIC.

Those connections at the INIC that transport Streaming Data serially towards the periphery (out of the INIC) are called SX0 and SX1. From the perspective of the application, they are the sources of the Streaming Data. Accordingly, there are the data sinks SR0 and SR1, which transport data from the periphery into the INIC.

10.9 SPI Port

The SPI port of the OS81110 is implemented as shown in figure 10.15. It can operate at a clock rate of up to 25 MHz (SCLKn = 512*Fs @ 48kHz) as SPI slave.

The INIC receives information about the direction of a data transfer from the SPI bus master through SDIN.

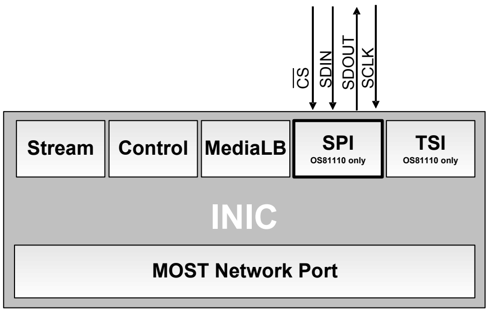


Fig. 10.15: INIC SPI Port

10.10 Transport Stream Interface (TSI)

The OS81110 has a serial 4-pin TSI interface capable of operating at bit rates up to 50 Mbps (1024*Fs). It supports TSI Master mode (transmission) or TSI Slave mode (reception).

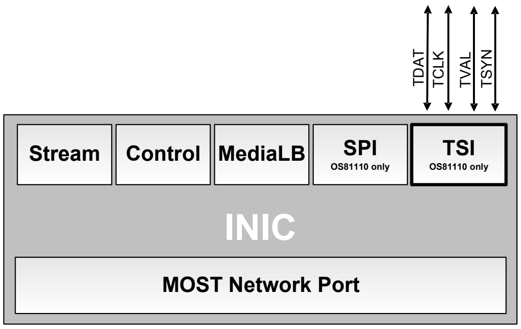


Fig. 10.16: INIC TSI Port

The TSI port packet structure uses a packet size of 188 Bytes. OS81110 has up to two 2 TSI ports that can run simultaneously.

10.11 MediaLB Port

The Media Local Bus (MediaLB) was developed to be able to transmit all types of multimedia data and additionally transmit data for controlling applications (see the most current MediaLB specification) [MediaLB]. These properties make it different from the generally used and highly specialized serial interfaces. It is a transmission medium which is synchronous to the MOST bus. It is used on printed circuit boards to connect different integrated components in a way that saves board space and keeps pin-count low.

MediaLB Signal Names	*MediaLB Physical Interface*		
	3-pin (single-ended)	*5-pin (single-ended)*	*6-pin (differential)*
MediaLB Clock (MLBC)	MLBCLK	MLBCLK	MLBCP MLBCN
MediaLB Signal (MLBS)	MLBSIG	MLBSI MLBSO	MLBSP MLBSN
MediaLB Data (MLBD)	MLBDAT	MLBDI MLBDO	MLBDP MLBDN

Table 10.2: Nomenclature of MediaLB Signals

MediaLB can be implemented with 3 pins, 5 pins, or 6 pins. The 5-pin MediaLB was specified to enable such components to be equipped with MediaLB, which have unidirectional pin circuits only. This results, however, in more space being required, additional logic circuits being used and the maximum speed being lower. In practice, the 3-pin model is thus mostly used, followed by the 6-pin MediaLB. MediaLB uses the following signals. For detailed naming information see Table 10.2:

- MLBCLK (MediaLB Clock, uni-directional)
- MLBSIG (MediaLB Signal, bi-directional)
- MLBDAT (MediaLB Data, bi-directional)

MLBC provides the timing for the entire MediaLB system. It is created by the MediaLB Controller and supports multiple clock rates. MLBC is synchronous to the MOST network and an input for all MediaLB devices. This principle is valid for all MediaLB implementations. Figure 10.17 shows the wiring for a 3-pin MediaLB implementation. For MediaLB 6-pin, there are modes which internally multiply the frequency of the MLBC signal by means of a PLL, so that the data transfer actually is based on an internally "recovered" clock that runs at higher speed. So there is a difference between the external and the actually used clock.

Actual MLBCLK	*MediaLB*			*Quadlets per Frame*	*Physical Channels*	*Physical Channels available for the application*
	3-pin	*5-pin*	*6-pin*			
256*Fs	X	X		8	PC0 - PC7	7
512*Fs	X	X		16	PC0 - PC15	15
1,024*Fs	X			32	PC0 - PC31	31
2,048*Fs			X	58	PC0 - PC57	57
3,072*Fs			X	87	PC0 - PC86	86
4096*Fs			X	117	PC0 - PC116	116
6144*Fs			X	161	PC0 - PC160	160
8192*Fs			X	215	PC0 - PC214	214

Table 10.3: Number of channels depending on the clock rate on MLBC

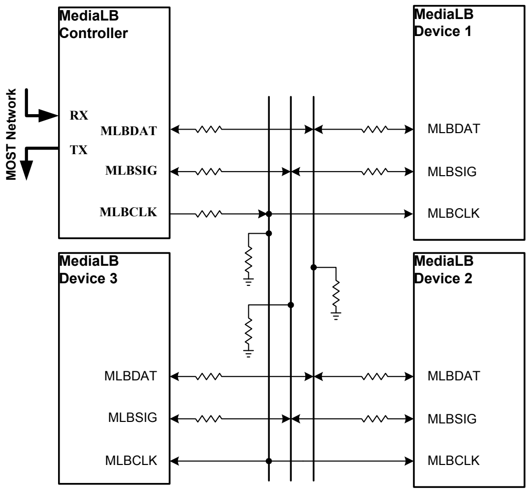


Fig. 10.17: Connection diagram of a 3-pin MediaLB implementation

In a MediaLB Bus, there is a MediaLB Controller, which is responsible for generating the clock. The MediaLB devices operate their MediaLB interface according to this predetermined clock.

Depending on the clock rate, there are different transport capacities available on the MediaLB (see Table 10.3). Four subsequent bytes (4 bytes = quadlet) are combined to form a physical channel. The first Physical Channel (PC0) of a MediaLB Frame is reserved for the System Channel, as it is called.

One or more Physical Channels can be combined to form a Logical Channel having a specified, unambiguous ChannelAddress. Logical channels operate unidirectional and transport only one type of data, which is selectable.

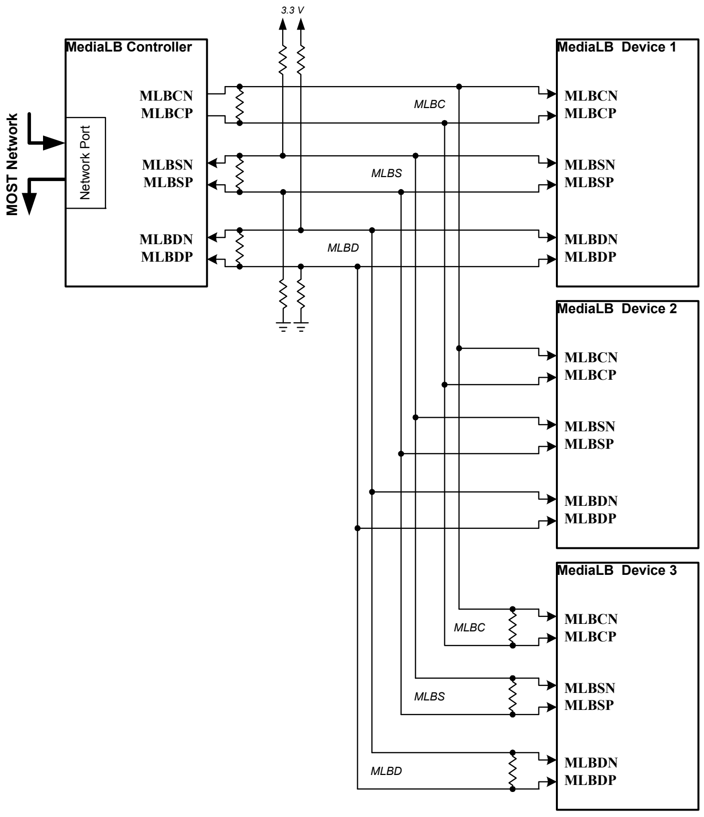


Fig. 10.18: Connection diagram of a 6-pin MediaLB implementation

MediaLB uses a time-multiplexed bi-directional signal line for ChannelAddresses, Commands and RxStatus signaling. This line is called MLBS. Opposite to the MLBC, it can be read and driven by all MediaLB devices. ChannelAddresses indicate which device can transmit data when and which devices can receive data on a particular logical channel.

The MLBD signal is the time-multiplexed bi-directional data line, which also can be driven or read by all devices. At a given time, only one device is allowed to drive this line. It is possible that more than one device is listening to MLBD.

Due to the fact that the MediaLB runs synchronously to the MOST network, the MediaLB is also organized in frames.

The control of the bus access is achieved on the level of the logical channels via the channel addresses. The MediaLB controller arbitrates data transfer amongst all MediaLB devices. The actual access is then carried out at a distance of one quadlet. It is used, e.g., by a MediaLB Device for sending out streaming data.

The separation into frames is done by means of a specific identification, the FRAMESYNC pattern. This pattern is also generated by the MediaLB Controller and signals that after one quadlet, the next frame with its system channel will start (see fig. 10.20).

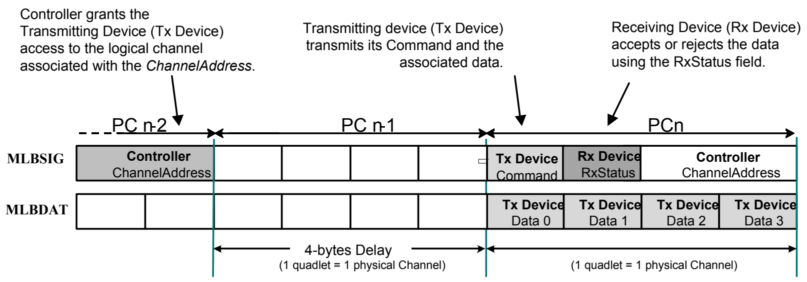

Controller grants the Transmitting Device (Tx Device) access to the logical channel associated with the ChannelAddress.
Transmitting device (Tx Device) transmits its Command and the associated data.
Receiving Device (Rx Device) accepts or rejects the data using the RxStatus field.
PC n-2
PC n-1
PCn
MLBSIG
MLBDAT
Controller ChannelAddress
Tx Device Command
Rx Device RxStatus
Controller ChannelAddress
Tx Device Data 0
Tx Device Data 1
Tx Device Data 2
Tx Device Data 3
4-bytes Delay
(1 quadlet = 1 physical Channel)
(1 quadlet = 1 physical Channel)

Fig. 10.19: Block Data transmission on the MediaLB

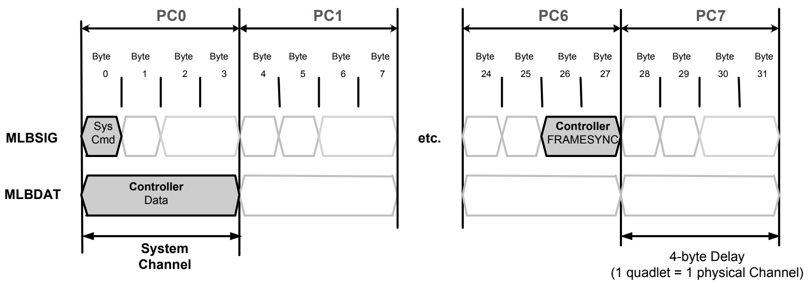


Fig. 10.20: MediaLB frame and the FRAMESYNC pattern (at 256*Fs)

The extended clocking options for MediaLB 6-pin and the design of the signal lines significantly increases data throughput on MediaLB. Further information can be found in the MediaLB Specification.

10.12 Power Management

The INIC supports external power management circuits by means of two separate input pins. Depending on the state of these pins, the INIC reacts according to the MOST Specification.

PS1	*PS0*	*Status*
0	0	U_{Normal}
0	1	Switch-To-Power (STP)
1	0	$U_{Critical}$
1	1	U_{Low}

Table 10.4: Input Pins for external Power Management

10.13 Intelligent Muting

Streaming connections in a MOST network are established between the devices as needed and usually stay alive as required. Nevertheless, there are imponderabilities that might lead to disturbances in the network and to unwanted audible or visible effects. The built-in intelligent muting of the INIC allows to automatically mute sink socket connections to prevent the application from receiving corrupted streaming data. Intelligent muting is triggered by either an unlock, or on occurrence of a Network Change Event (NCE, related to an MPR change - for instance when one node has left the ring).

Automatic muting is available for synchronous data, and DiscreteFrame Isochronous Streaming data (IsocDiscFrameData) on both, the Streaming Port and the MediaLB Port.

10.14 Network Interface Controller (NIC)

10.14.1 Available Interfaces

In contrast to INICs, Network Interface Controllers (NICs) are controlled by direct read/write access to individual memory locations. The roles of the existing interfaces therefore have a different definition than in the case of INICs. The application running on the EHC is responsible for the behavior of the entire device, e.g., for start-up and error management.

This resulted in a few obstacles for the MOST systems of the first generation, which were taken into account when developing the INIC. One of these obstacles was the delayed start-up of the application of some devices. Due to the bypass being closed for too long, there was a risk of delayed system starts or of instabilities because of missing data recovery. Errors in applications could trigger errors in the recording of storage positions in the NIC. This in turn had effects on the entire MOST network.

These weaknesses were addressed by encapsulating these mechanisms within the INIC. At the same time, the Media Local Bus (MediaLB) replaced the available 8-bit interface of the NIC, which resulted in less space requirements and less implementation expenses. This is why new developments are realized with the INIC.

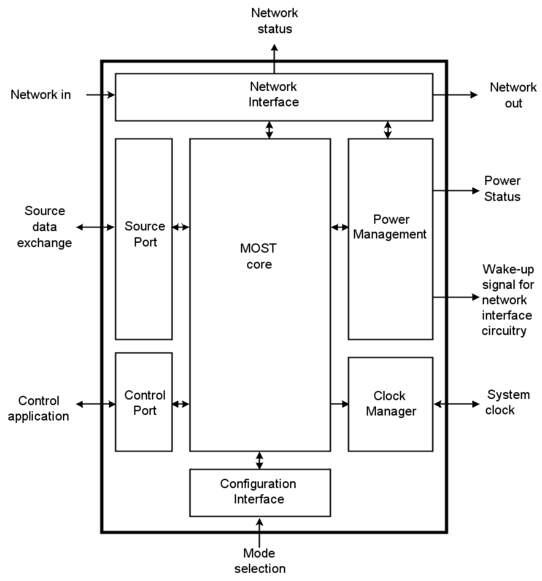


Fig. 10.21: Block diagram of an OS8104A

This section takes the NIC component OS8104A as an example. It has meanwhile replaced the NIC of the first generation (OS8104) when updating existing designs.

The basic configuration of the OS8104A is done during the hardware initialization, via the Configuration Interface (CI).

The Control Port (CP) allows write/read access to the internal memory of the NIC. It thus enables fine tuning configuration of the component, but also communication via Control Messages and packet data transfer up to 48 bytes packet length. Also routing, i.e., processing of the streaming data, is controlled via CP. It can be operated in serial or in parallel.

It is the main task of the Source Port (SP) to transport streaming data. As the hardware of the SP is generally able to handle very high data rates, there are further modes for the SP in parallel operation:

- Parallel-Asynchronous mode
- Parallel-Synchronous mode
- Physical (Parallel-Combined) mode

10.14.2 Serial Operation

CP

In serial operation, the CP can transmit data either in the I²C format (as I²C-Slave with Clock Stretching) or in the SPI format.

SP

Similar to the INIC, the data signals of the Source Port share a Clock Pin (SCK) and a Synchronization Pin (FSY, for distinguishing left/right). If FSY and SCK are configured as outputs, they drive the serial data transmission on the Streaming Port. In this case, the externally connected peripheral components must synchronize to FSY and SCK. If this is not possible, FSY and SCK can be configured as inputs.

10.14.3 Parallel Operation of CP and SP

Note: If FSY and SCK are configured as inputs, the synchronization of the application must take place via the RMCK (Recovered Master Clock) output of the NIC.

CP

In parallel operation, data is written to or read from the memory of the NIC in a byte-wise manner. The parallel operation requires additional control signals, which are described below.

SP

The parallel operation significantly increases the data throughput of the SP. Performance is so high that a single component per MOST Frame can simultaneously read the complete 60 bytes streaming data from the MOST network and write back 60 new bytes. In the Physical Mode, 64 bytes per frame can flow from the application to the MOST network and back. In this case, additional control information for the interface operation is included.

10.14.4 Possible Configurations (Serial/Parallel)

It is possible to operate the NIC in a completely serial or completely parallel manner. There is also a combination of serial and parallel operation, in which the Control Port is operated in the serial mode and the Source Port already in the parallel mode.

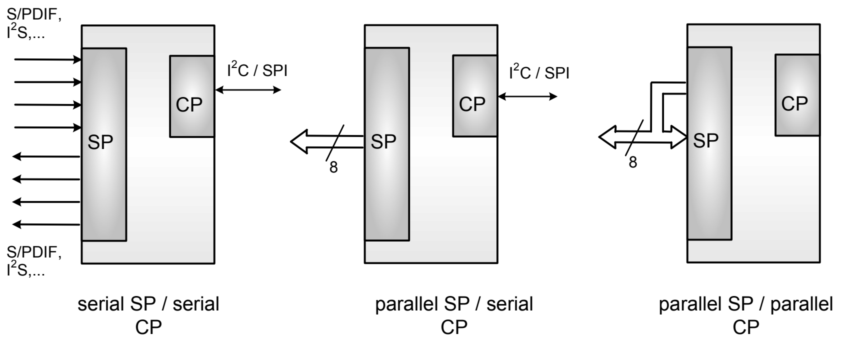


Fig. 10.22: Possible basic configurations for CP and SP

Note: If parallel operation is selected, the serial Source Port pins (SX0 to SX3, and SR0 to SR3) are no longer available. They are converted to an 8-bit wide data bus.

10.14.5 The Basic Configuration of a NIC

The basic configurations for parallel or serial operation are done during the hardware initialization. The Configuration Interface, as it is called (a group of connection pins), provides this in cooperation with the Reset Pin of the NIC. It consists of the following pins:

- PAR_CP (Control Port parallel)
- PAR_SRC (Source Port parallel)
- ASYNC (parallel asynchronous mode of the SP)
- LEG (OS8104 compatibility mode)
- SCL (Clock of the serial CP interfaces)

These pins are read with the rising edge of the Reset Pin. The NIC then carries out the relevant internal configuration. It signals the end of the configuration and thus readiness for operation by means of triggering off the first interrupt (Power-On Interrupt).

Note: Before the first access to the CP, the application must, by all means, wait for the Power-On Interrupt.

After that, none of the pins – except the SCL – must change their mode. SCL is a Clock Pin which is used in the serial CP Interface. Depending on the selected bus, this pin is either at logic “1” (I^2C) or at logic “0” (SPI) in idle mode. It is assumed that in the case of the Reset there are no data transmissions into the CP, i.e., there is an idle phase, in order to enable proper detection.

10.14.6 Interfaces for Control Data in NICs

Control data is always fed into the component via the CP and read from there, independent of the CP being operated in parallel or serial. The Control Messaging is based on internal buffers of the NIC, i.e., a group of correlated registers.

10.14.7 Interfaces for Streaming Data in NICs

The SP provides for a multitude of streaming formats and data rates in both, the serial and the parallel mode. Control of the flow of Streaming Data is ensured via the MOST Routing Table (MRT). This table controls the Routing Engine, which carries out the actual data transport.

In the serial mode, eight serial Interface Pins are available at the maximum. Depending on format and speed, they can all operate simultaneously. There are four pins in each direction (in/out) to which A/D converters, codecs or DSPs can be connected.

The highest throughput of Streaming Data is obtained via the Parallel Synchronous Mode of the SP. The Physical Mode is based on the same Interface Mode. It has the same throughput with regard to the Streaming Data.

10.14.8 Interfaces for Packet Data in NICs

Packet data can be sent and received via the internal memory of the NIC. There, data is accessible either via the CP, or the SP in the Parallel Asynchronous Mode. The maximum payload is 48 bytes in this case.

If longer packet lengths and thus higher data throughputs are required, the Physical Mode (Parallel Combined Mode) can be used. This interface requires an appropriately dimensioned connection to the application.

11 Tools

11.1 Overview

The development process of MOST systems uses the well-known V-Model. The left side of the V-Model, as illustrated in figure 11.1, is composed of three phases. In the market, there are same simulation and analysis tools that represent these phases.

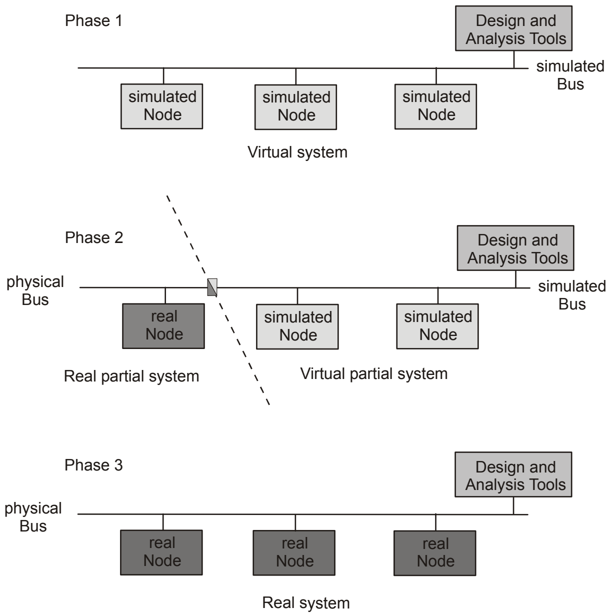


Fig. 11.1: The development process from network simulation to the real total system (according to [vector])

In phase 1, the simulation of the complete system with respect to the requirements takes place. Here, the functional and communicated behaviors of the MOST nodes will be verified. In this phase, the function blocks are defined by a MOST editor and the dynamical behavior is described by an MSC editor (refer to section 11.2).

During phase 2, the MOST ECUs are developed and integrated step by step into the network. There are real and virtual ECUs in this phase, which is called Remaining Bus Simulation as well. Rapid prototyping tools drive push on this process (refer to section 11.4). Software libraries for the Network Service (refer to chapter 9), which are promoted by several software houses, support the software development. INIC Remote Viewer and MediaLB Analyzer provide the hardware development (refer to section 11.5).

In phase 3, all real ECUs are available. The complete system can be tested and verified now.

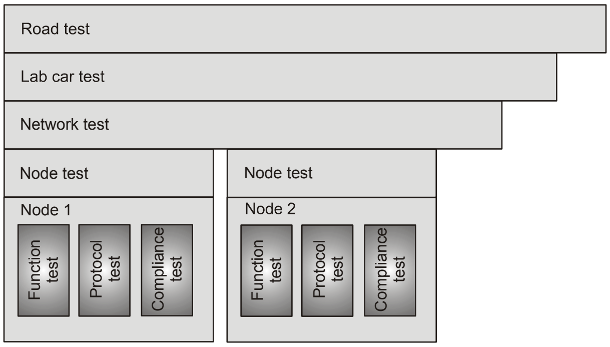


Fig. 11.2: Overview over the test levels of a MOST system

Tests of the single MOST ECUs and the complete network take place in phases 2 and 3. Figure 11.2 illustrates the necessary test levels. At first, function and compliance tests are executed of a single ECU. The function tests warrant the accurate implementation of the user application, and the compliance and protocol tests check the correct communication behavior in regard to the MOST Specification. Following, the complete MOST network is tested in the lab, then on the test board and finally in a car. Chapter 12 introduces the compliance tests and chapter 13 the MOST tests.

11.2 Specification Tools

11.2.1 MSC Editor

The Message Sequence Chart Editor (MSC editor) is the graphical tool that is used to edit Message Sequence Charts (MSCs). More advanced tools are visual modelling tool for analysis, specification, documentation and testing of communications services and protocols. There are open source and commercial MSC editors.

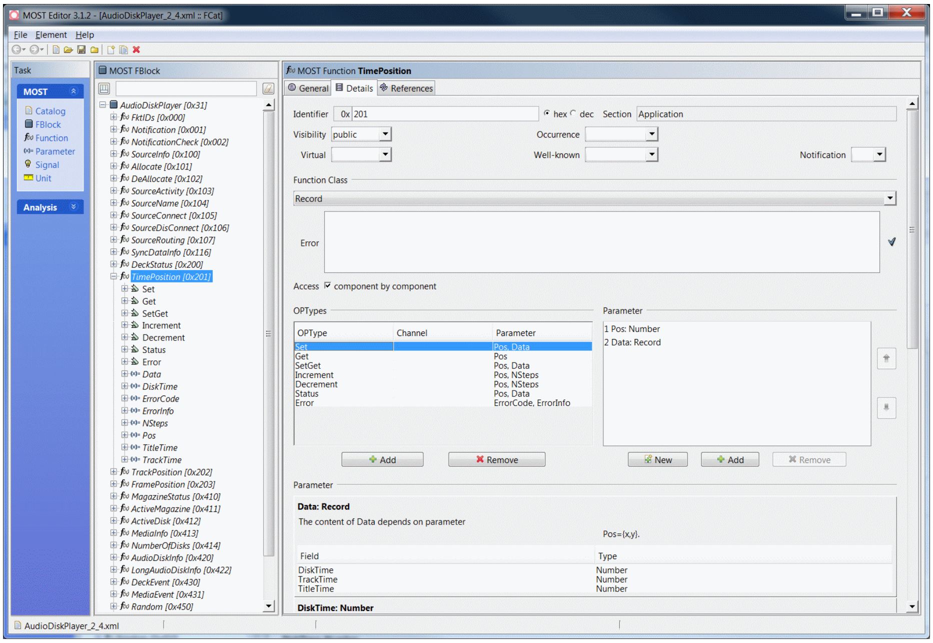
MSCs are described in appendix A.

11.2.2 MOST Editor

Some OEMs maintain a database that contains the complete communications parameters of all bus systems of a car, including the MOST function blocks. They use proprietary editing tools in this case. However, the MOST Cooperation has developed its own MOST Editor, which is described in this section.

The MOST Editor can be used for creation and data management of function catalogs, also provided by the MOST Cooperation, in XML format. The function catalog contains the function blocks with all their defined methods and properties. The individual description of the MOST device is based on this catalog. Those methods and properties of the function block that are not to be implemented are deleted and the value range of the existing parameters is adapted.

The actual user interface is divided into two parts (see fig. 11.3). The left side contains the function tree. It shows the individual function blocks with their functions. The right side contains a detailed view of the functions.



AudioDiskPlayer [0x31]
- FktIDs [0x000]
- Notification [0x001]
- NotificationCheck [0x002]
- SourceInfo [0x100]
- Allocate [0x101]
- DeAllocate [0x102]
- SourceActivity [0x103]
- SourceName [0x104]
- SourceConnect [0x105]
- SourceDisConnect [0x106]
- SourceRouting [0x107]
- SyncDataInfo [0x116]
- DeckStatus [0x200]
- TimePosition [0x201]
  - Set
  - Get
  - SetGet
  - Increment
  - Decrement
  - Status
  - Error
  - Data
  - DiskTime
  - ErrorCode
  - ErrorInfo
  - NSteps
  - Pos
  - TitleTime
  - TrackTime
- TrackPosition [0x202]
- FramePosition [0x203]
- MagazineStatus [0x410]
- ActiveMagazine [0x411]
- ActiveDisk [0x412]
- MediaInfo [0x413]
- NumberOfDisks [0x414]
- AudioDiskInfo [0x420]
- LongAudioDiskInfo [0x422]
- DeckEvent [0x430]
- MediaEvent [0x431]
- Random [0x450]

Identifier 0x 201 [✓ hex] dec Section Application

Visibility public Occurrence

Virtual Well-known Notification

Function Class

Record

Error

Access ✓ component by component

OPTypes

OPType	Channel	Parameter
Set		Pos, Data
Get		Pos
SetGet		Pos, Data
Increment		Pos, NSteps
Decrement		Pos, NSteps
Status		Pos, Data
Error		ErrorCode, ErrorInfo

Parameter

1 Pos: Number
2 Data: Record

Parameter

Data: Record

The content of Data depends on parameter

Pos={x,y}.

Field	Type
DiskTime	Number
TrackTime	Number
TitleTime	Number

DiskTime: Number

Fig. 11.3: MOST Editor

The function tree can be extended or amended using the context menu that can be activated by clicking the respective entry with the right mouse button. After selecting a function, its properties are shown in the left window.

The description of a function must have the following entries:

- ID
- Function class
- Section
- Textual description in HTML format
- Available OPTypes (e.g., Get, Set)
- Version information

Figure 11.3 shows the dialog for defining the operation types of a function. In the example given, the function *TimePosition* of the function block *AudioDiskPlayer* has the OPTypes *Get*, *Set*, *SetGet*, *Increment*, *Decrement*, *Status* and *Error*. In the next step the parameters for each OPType are defined.

All parameters are stored in the XML function catalog containing the following entries [Cat 02]:

- XML version
- Function catalog header
- Header of the current function block
- List of the functions with their parameters
- Definitions of the function classes

The following listing shows the basic structure of the function catalog:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE FunctionCatalog (View Source for full doctype...)>
<FunctionCatalog>
  <CatalogVersion>
  </CatalogVersion>

  <!-- FBlock: AmFmTuner -->
  <FBlock>
      <Function>
      ….
      </Function>

      <Function>
      ….
      </Function>
  </FBlock>

  <Definition>
  …
  </Definition>

</FunctionCatalog>
```

11.3 Simulation and Analysis Tools

Simulation and analysis tools are a set of tools for the development, testing and analysis of MOST networks using a consistent Graphical User Interface (GUI). At the beginning of the development process, they are used to create simulation models that emulate the behavior of the MOST nodes. During the development, these models serve as a foundation for analysis, testing and integration. The tools should support the following functions:

- **Online analysis** of MOST network data enables data analysis in real time. **Offline analysis**, on the other hand, evaluates recorded data. Specific events can be highlighted or scoped based on the adjustments of the filter and trigger settings. A time stamp synchronization of Viewer, Graph and Watch functions supports the evaluation of data in chronological alignment.
- A **stress tool** generates load on the Control and Packet Data Channel and applies modulated signal & lock sequences with precise timing.
- A tool for the **access to the hardware and network states** such as light & lock, system lock and configuration status. Together with the stress tool, it gives a statistic overview of the quality of transmission.
- A **Viewer for the Control Channel** enables the analysis of CMS and AMS protocols on a high level. It shows MOST events on the MOST ring.
- An **FBlock Viewer** gives an overview of the application states that are communicated over the bus. It shows all or selected properties values transmitted over the bus. It needs the XML function catalog for the disassembling of the values.
- A **Viewer for the Packet Data Channel** should show the MOST High Protocol, MAMAC and MDP, depending on which of those are implemented. In MOST150 systems, it should visualize the Ethernet channel. A transmitter function sends packet data to other network nodes.
- The tools should enable the **access to the Streaming Channel**. All synchronous and isochronous channels may be transmitted with 44.1 kHz or 48 kHz to the PC or in the reverse direction from the PC onto the ring.
- If the tool analyzes **MOST slave nodes**, it has to emulate TimingMaster, NetworkMaster and PowerMaster
- The Central Registry stores all FBlocks that are available in the MOST system with node position address and logical address as well as the instance ID. A viewer can show the FBlocks with the parameters and enable a comfortable access to the system.

11.4 Rapid Prototyping Tools

Rapid Prototyping Tools drive the development of MOST ECUs. You can choose between the following types:

- Network Services Library; It supports the software development and was considered in chapter 9.
- Plug-in for Simulation and Analysis Tools
- Hardware platform Rapid Prototyping and Evaluation
- PC Tool Kit
- System components

11.4.1 Plug-in for Simulation and Analysis Tools

These tools must provide a quick way to create control panels for MOST devices. The panels serve to display bus signals in user-defined windows with graphic controls. Often, various controls are available such as text boxes, sliders, and pointer instruments. The panel design may include static elements (text or frames), macros as well as background graphics. Scripting languages enable the automatic execution of simulation, test and analysis functions.

11.4.2 Hardware platform for Rapid Prototyping and Evaluation

The hardware platform consists of the physical layer (oPhy or ePhy), an INIC for MOST25, 50 or 150 and offers interfaces to the External Host Controller (EHC) and to the audio/video in- and outputs. Figure 11.4 shows the fundamental architecture of the platform. A Microcontroller or FPGA on board separates the Streaming and Packet Data Channel as well as the Control Channel. It transfers the Packet Data Channel and the Control Channel to a PC via USB and the Streaming Data to a codec. On the PC, the Network Service and an EHC application are running. A MediaLB header could provide an interface for the MediaLB analyzer.

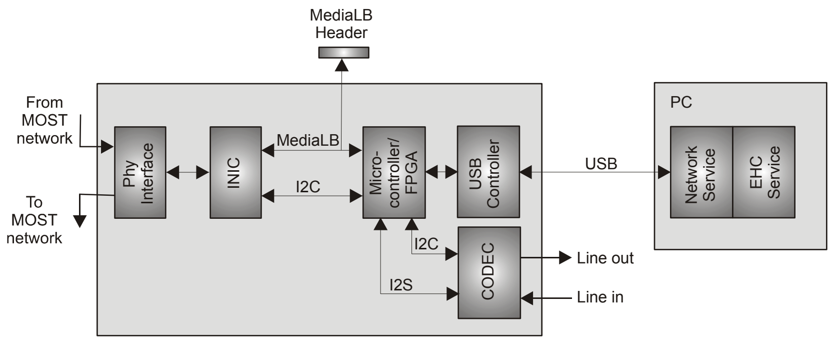


Fig. 11.4: Architecture of a platform for rapid prototyping and evaluation

11.4.3 PC Tool Kit

PC tool kits often contain two components: a MOST Interface as well as a bundle of multimedia applications and MOST control functions. Figure 11.5 shows the architecture. The MOST Interface provides a high-speed connection between a PC and the MOST network. It allows access to the Packet and the Streaming Data Channel as well as the Control Channel and consists of the physical layer (ePhy or oPhy), an INIC and the firmware. The interface to the PC could be PCI, USB or PXI.

The tool kits allow software development for devices and own tools without any additional hardware.

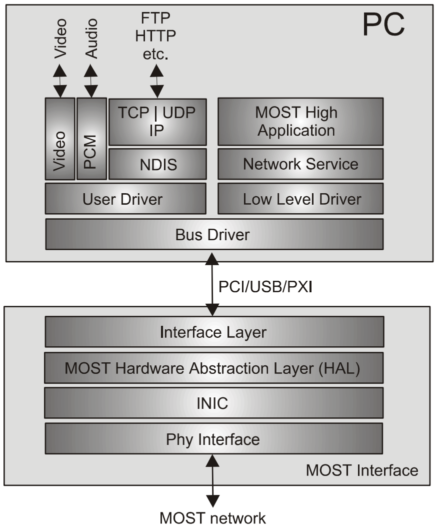


Fig. 11.5: Architecture of PC tool kits

11.4.4 System components

For an effective system development, repeaters, hubs, gateways, and converters can be used.

The repeater is a device that can act as a network extender or provide monitor-out functionality without interfering with the network's structure. A hub recovers the signals and distributes data streams in a star-like architecture. A gateways provides interconnection from MOST to other networks such as CAN.

A converter converts network signals between different physical layers, e.g., between MOST50 electrical physical layer (ePhy) and the optical physical layer (oPhy). It is used to validate electromagnetic compatibility (EMC) and operational electrostatic discharge (ESD) behaviors of a MOST50 ePhy Device Under Test (DUT).

11.5 Hardware Development Tools

Each MOST ECU has got an INIC (older versions have a NIC) that can be analyzed with two specialized tools. Those are the INIC Remote Viewer and the MediaLB Analyzer.

11.5.1 INIC Remote Viewer

INIC Remote Viewer provides a detached gateway which is used to remotely analyze INIC-based MOST network node information. The viewer can provide the following functions:

- Monitors properties and states of FBlock NetBlock and FBlock INIC, usually only accessible through the Debug Header
- Reads the INIC Configuration String
- Visualization of:
    - INIC parameters
    - NetBlock parameters
    - INIC states and state changes
    - Ports, sockets and connections
    - Configuration string entries

11.5.2 MediaLB Analyzer

The MediaLB Analyzer allows observation and visualization of MediaLB data and provides following functions:

- Disassembly of INIC Port Messages
- Filter and trigger definitions
- MediaLB lock and speed detection
- Speed modes for MediaLB 3-Pin and 6-Pin
- MediaLB frame and physical channel counter
- MediaLB link layer protocol check

12 Compliance Tests

12.1 Objectives of Compliance Tests

As already explained in chapter 11, the integration of network systems and modules is particularly challenging due to their complexity. An effective measure for meeting the challenge is to introduce a certification test process (Compliance Process) so that the devices and modules are tested before they are integrated into the system. This considerably increases the quality and reduces the complexity of integration.

It is the primary aim of the MOST Certification Test Process (Compliance Process) to test the modules and devices with regard to their conformity with the specification [ISO 9646] [ETS 300 406], i.e., whether the requirements of the specification were met. The aspect of quality is closely related to the aim of reaching a high level of interoperability.

The Compliance Tests cover a legal aspect within the MOST Cooperation. The tested device is only allowed to participate in the MOST IP (Intellectual Property) pool and to use the MOST® trademark (see fig. 12.1) after the compliance tests are passed.

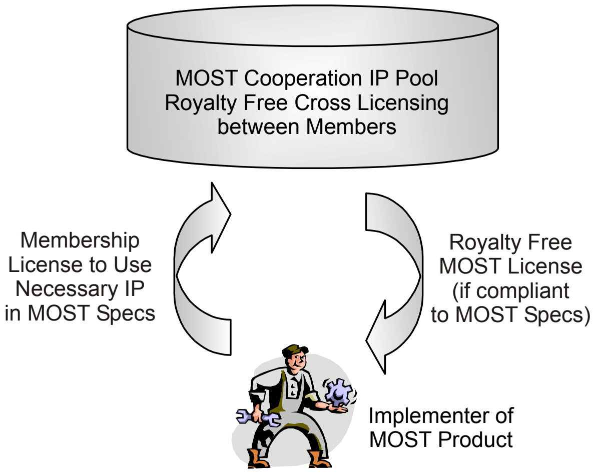


Fig. 12.1: MOST IP and Compliance Process

In accordance with the layer model, the Certification Test Process covers the following areas (Compliance Levels) (see fig.12.2):

- Module tests on the Physical Layer (e.g. FOT, Pigtail)
  (Full Physical Compliance)

- Device test on the Physical Layer
  (Limited Physical Compliance)
- Device test on higher communication levels
  (Core Compliance)
- Device test on the Application Layer
  (Profile Compliance)

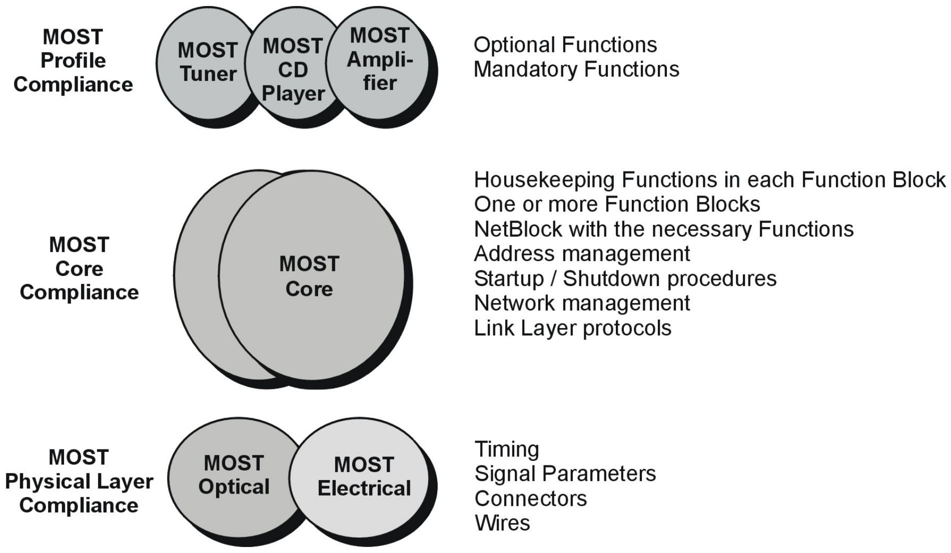


Fig. 12.2: Compliance Levels

12.2 Organization of the Compliance-Process

12.2.1 Authorities for the Compliance Process

Several authorities were installed to work together on the Certification Test Process within the MOST Cooperation (see fig. 12.3). The MOST Compliance Supervisory Board (MCSB) is responsible for controlling and supervising the entire process. It authorizes the MOST Compliance Administrator (MCA) to supervise the Test Process. He accepts the test reports from the MOST Compliance Test Houses, checks them and finally grants the certificate.

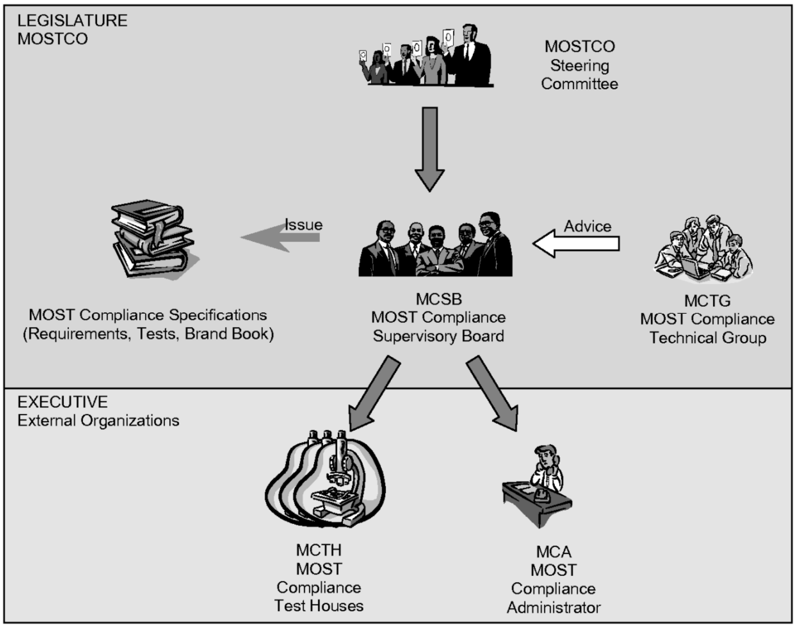

LEGISLATURE
MOSTCO
MOSTCO Steering Committee
Issue
Advice
MOST Compliance Specifications (Requirements, Tests, Brand Book)
MCSB MOST Compliance Supervisory Board
MCTG MOST Compliance Technical Group
EXECUTIVE
External Organizations
MCTH MOST Compliance Test Houses
MCA MOST Compliance Administrator

Fig. 12.3: Organization of the Compliance Process (source: [ComReq])

The performance of the Compliance Test is effected by MOST Compliance Test Houses (MCTHs) approved by the MOST Cooperation. They are accredited according to the ISO 17025 *General Requirements for the Competence of Testing and Calibration Laboratories* by the German Accreditation Body for Engineering (DAkkS- Deutsche Akkreditierungsstelle GmbH) in Berlin /Frankfurt, Germany.

The test specifications, errata, as well as the technical consulting are effected by the MOST Compliance Technical Group (MCTG) – a board of experts in which the test houses are represented.

12.2.2 Workflow of the Compliance Process

The workflow of the certification test is carried out in three steps:

Step 1
The manufacturer selects a MOST Compliance Test House (MCTH) and develops the test plan together with the test house. For that purpose, the manufacturer has to make all documents required for the test available to the MCTH.

Step 2
The MCTH carries out all necessary tests and writes the test report as well as manufacturer's declaration, the Declaration of Compliance (DoC), which is to be added to the Compliance File.

Step 3
The MOST Compliance Administrator (MCA) checks the Compliance File and lists the product, upon application by the manufacturer, as a MOST *Compliant* certified product in the MOST Compliance Product List (MCPL). Thus the manufacturer is granted the MOST license.

Figure 12.4 gives a survey of the workflow of the Compliance Process.

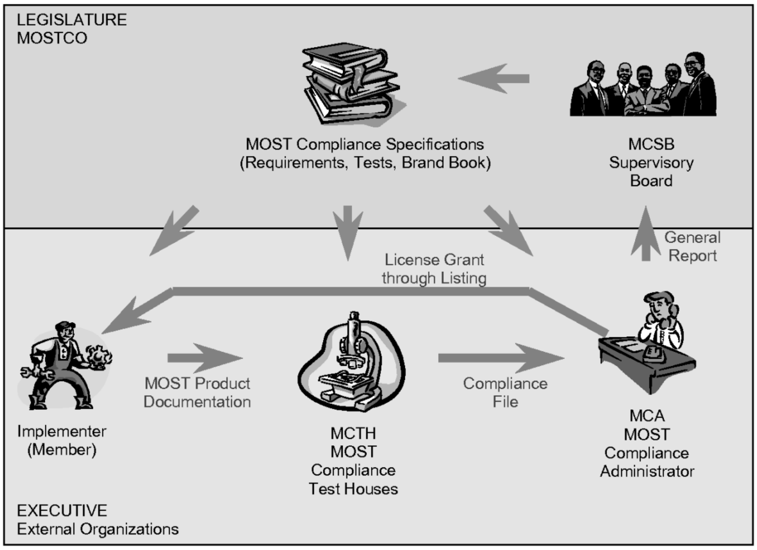

LEGISLATURE
MOSTCO
MOST Compliance Specifications (Requirements, Tests, Brand Book)
MCSB Supervisory Board
General Report
License Grant through Listing
Implementer (Member)
MOST Product Documentation
MCTH MOST Compliance Test Houses
Compliance File
MCA MOST Compliance Administrator
EXECUTIVE
External Organizations

Fig. 12.4: Workflow of the Compliance Process (source: [ComReq])

12.3 Physical Layer Compliance Tests

With regard to the Physical Layer Compliance, the measurement principles and the measurement methods were defined first, as well as the tolerances on which the tests were to be based. Within the scope of MOST Compliance, there are test points SP1 to SP4 (see fig. 12.5 and chapter 6).

In the second step of the Compliance Process the procedures to be carried out (test cases) were defined. They indicate voltage and temperature ranges for which the following measurements at test points SP1 to SP4 are to be carried out in particular:

- Bit rate
- Logical levels
- Rise times
- Pulse width variation and distortion

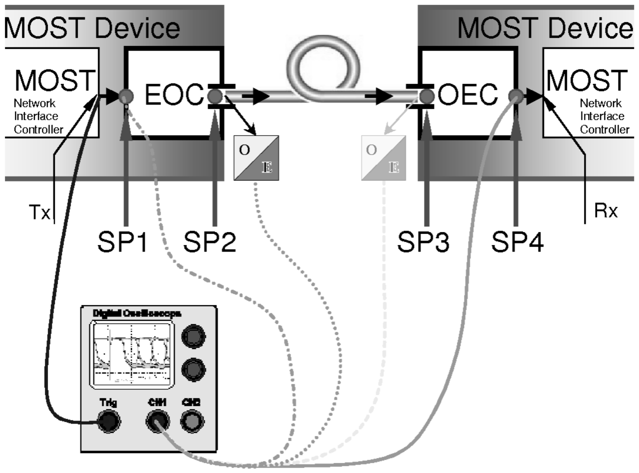


Fig. 12.5: Example of a test setup for the Physical Layer Compliance (EOC – Electrical/Optical-Converter; OEC – Optical/Electrical-Converter) (Source: [ComPhyLP 1.0])

- Data-dependent and uncorrelated jitter
- Input resistance and capacity
- Wave length determination
- Optical output power
- Extinction
- Positive and negative overshoot behavior
- Signal ripple

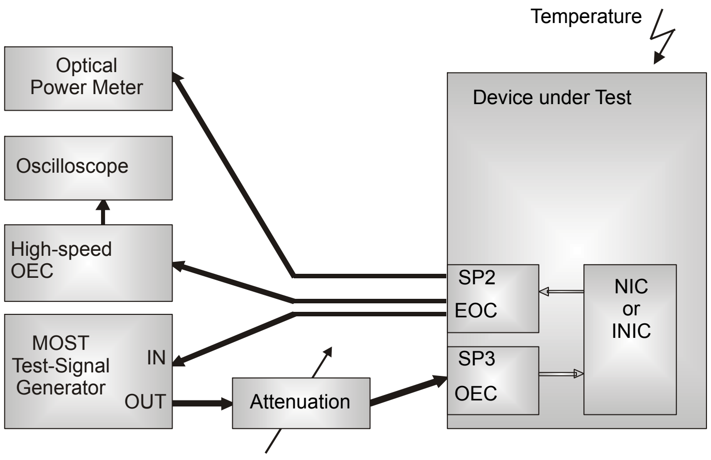


Fig. 12.6: Test setup for the Limited Physical Layer Compliance Test

The Limited Physical Layer Compliance Test aims at testing complete devices and can be considered as a Blackbox test, since only test points SP2 and SP3 can be tested. Figure 12.6 shows the setup for the Limited Physical Layer Compliance Test. For MOST150, the Physical Layer Stress Test Tool is applied.

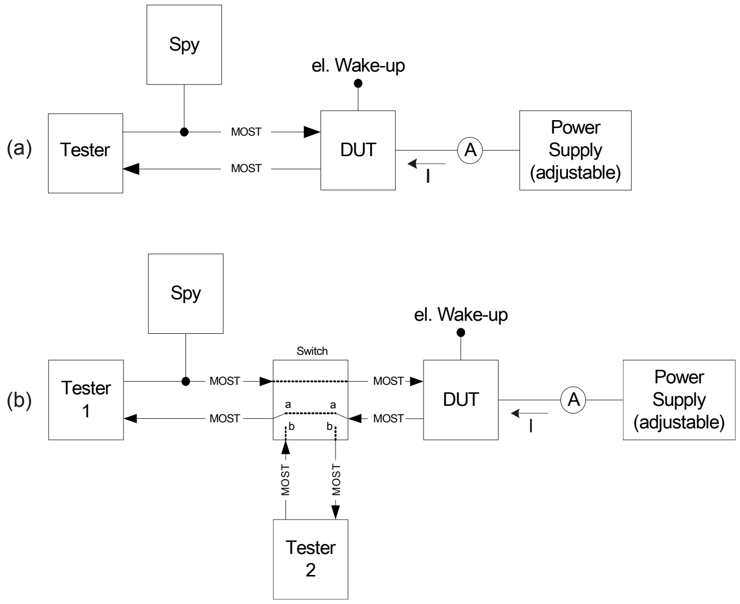


Fig. 12.7: Test setup with one (a) and two (b) testers for Core Compliance Tests

12.4 Core Compliance Tests

Based on an analysis of the MOST Specification, the requirements (specification points) for the Core Compliance Tests were determined and the test cases were written. These test cases verify the following behaviors:

- Wake-up behavior
  - Wake-up – TimingMaster
  - Wake-up – woken up TimingSlave
  - Wake-up – waking TimingSlave
- Normal behavior (Normal Operation)
- Power Management
- Error Management
- Ring Break Diagnosis

- System Configuration
  - Initialization (NetworkMaster)
  - System Configuration (NetworkMaster)
  - System Configuration (NetworkSlave)
- Node addressing
- Control-Messages (Ack/Nack-Test)
- Notification matrix
- Segmentation of messages

Figure 12.7 gives an overview of the necessary test setups. The upper part of the figure (a) shows the simple setup with one tester, the lower part (b) shows the setup with two testers, by means of which the behavior in the ring can be simulated:

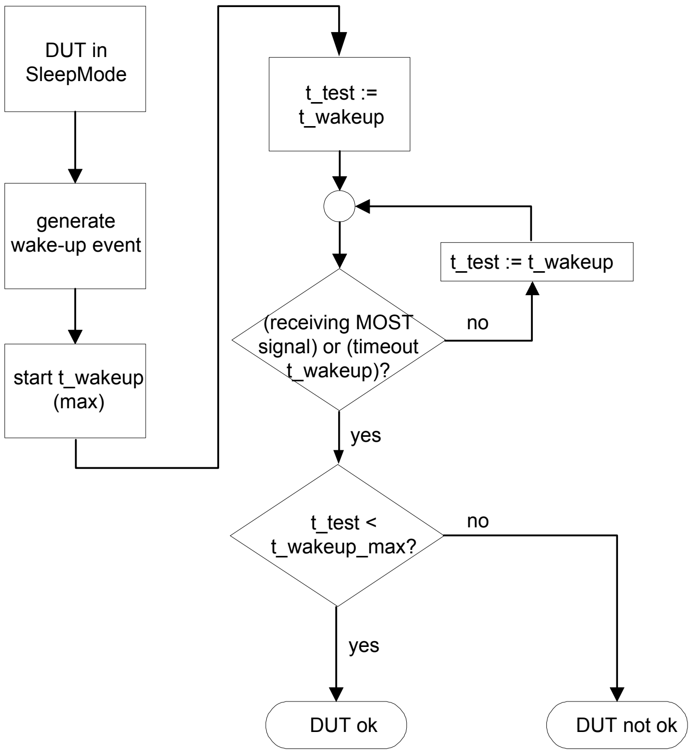


Fig. 12.8: Test case *Signal On Test*

It is often necessary to support the workflow of the tests with function block EnhancedTestability (ET), which acts as a "filter" between the application and the MOST Network Interface Controller and can be triggered by various events. It features the following characteristics:

- It is located between the application and the NetInterface, it is mandatory for each device or each MOST node and must not have any effect on the overall device behavior. This applies in particular if the device is not under test conditions.
- It generates or suppresses signals.
- The FBlock is only available in the NetOn state and does not store any values about a power failure.
- Typical functions of the FBlock ET are, for example, AutoWakeup, NetInterfaceState, SendMessage, EchoMessage or Reset.

Figure 12.8 shows an example of the workflow of the test case *Signal On Test*.

12.5 Profile Compliance Tests

Profile Compliance Tests cover, as seen in figure 12.2, the individual applications (profiles), such as the ConnectionMaster. Figure 12.9 shows the setup of the Profile Compliance Tests. They make use of the general structure of the profiles (function blocks). Since the FBlocks are derived from the generic FBlock template GeneralFBlock, the parts of the FBlocks that are to be tested generically are also described in a high level MOST Profile Compliance Test Specification.

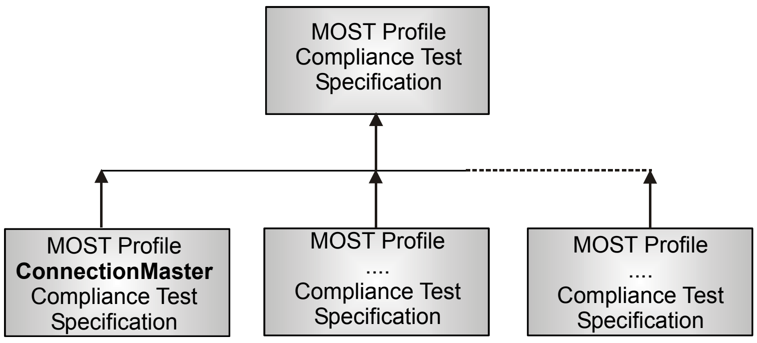

MOST Profile Compliance Test Specification
MOST Profile ConnectionMaster Compliance Test Specification
MOST Profile .... Compliance Test Specification
MOST Profile .... Compliance Test Specification

Fig. 12.9: Structure of the Profile Compliance

Distinctive features and the dynamic behavior, in particular, are tested in the profile-specific test specifications derived from the FBlocks.

Figure 12.10 shows the example of a Message Sequence Chart (MSC) used to describe a test case ("SyncConnectionTable_Get") for the Profile Compliance Test. The test cases specified in this manner serve as a basis for the MOST Compliance Test Houses (MCTH) and for the test tools described in section 12.7. The tests defined there, based on the abstract test specifications, must still be implemented.

12.6 Interoperability Tests

Interoperability Tests are defined as functional tests with respect to other functional MOST products. Interoperable MOST products in the field are meant to increase the user's trust in the MOST system. This is why the MOST Compliance Process calls for Interoperability Tests.

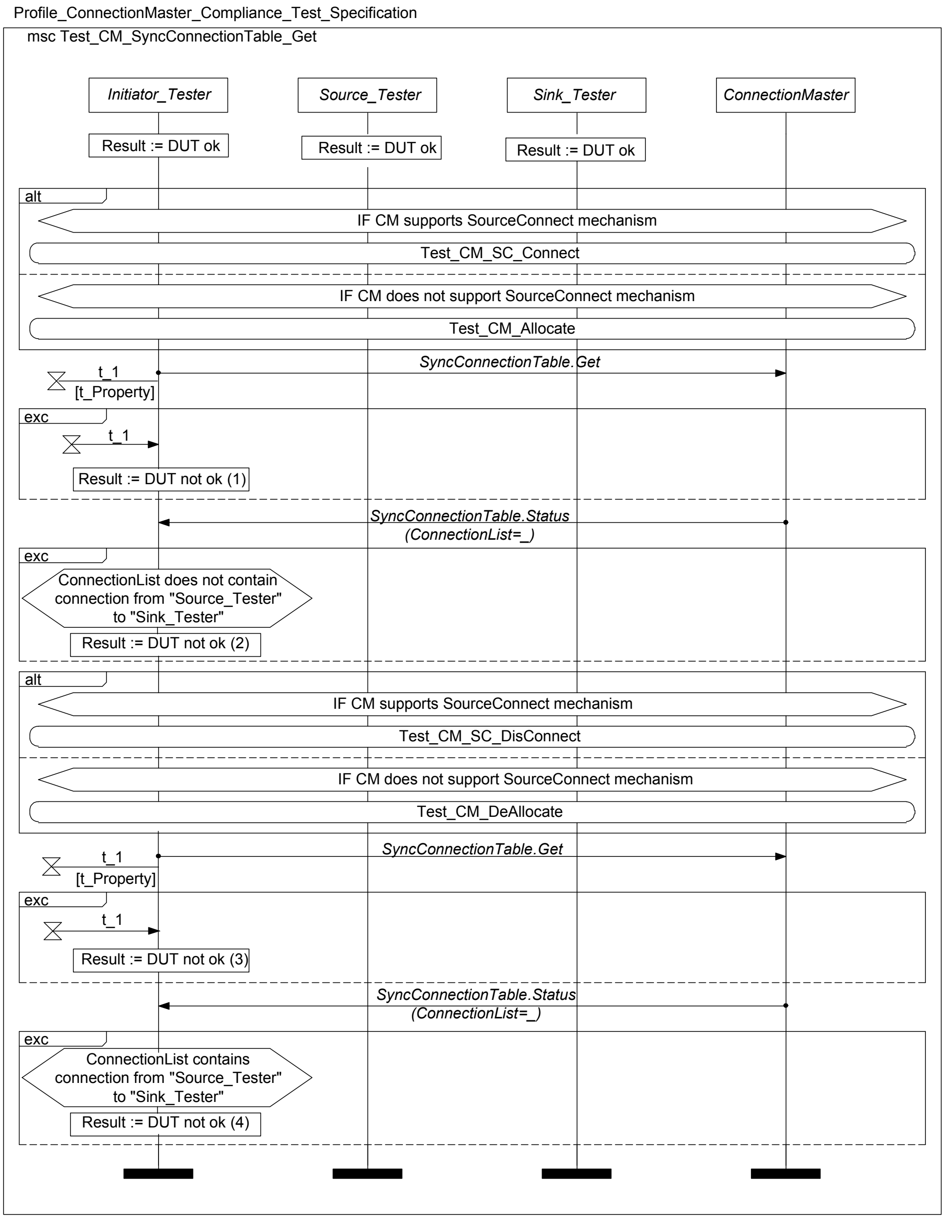

Profile_ConnectionMaster_Compliance_Test_Specification
msc Test_CM_SyncConnectionTable_Get
Initiator_Tester
Source_Tester
Sink_Tester
ConnectionMaster
Result := DUT ok
alt
IF CM supports SourceConnect mechanism
Test_CM_SC_Connect
IF CM does not support SourceConnect mechanism
Test_CM_Allocate
SyncConnectionTable.Get
t_1
[t_Property]
exc
Result := DUT not ok (1)
SyncConnectionTable.Status (ConnectionList=_)
ConnectionList does not contain connection from "Source_Tester" to "Sink_Tester"
Result := DUT not ok (2)
Test_CM_SC_DisConnect
Test_CM_DeAllocate
Result := DUT not ok (3)
ConnectionList contains connection from "Source_Tester" to "Sink_Tester"
Result := DUT not ok (4)

Fig. 12.10: Example of a Profile Compliance Test Case (SyncConnectionTable-Get)

In order to make sure that Interoperability Tests are reproducible and can be carried out consistently by all manufacturers, a set of products has been defined for conducting Interoperability Tests. These products are referred to as Golden MOST Implementations (GMI) and are registered in a corresponding list. The devices to be used for a test depend on the functional usability of the products.

13 Testing MOST based Infotainment Systems

13.1 Challenges in Testing MOST Systems

The integration of modern and powerful infotainment systems, which today are largely based on MOST, results in considerable challenges for the OEM which, in turn, influence the way in which the systems are tested. Typically, different suppliers deliver their respective MOST components to the OEM, where they are assembled into a complete system.

While developing the components, the suppliers employ various, in-house development processes, technologies and standards. The OEM then has the task of integrating the components and ensuring that they interact correctly within the context of the overall vehicle system. Some important system functions can only be achieved through the cooperation of a number of components. An example for such a function is accepting an incoming hands-free phone call while a music CD is being played: as soon as the phone call reaches the car, the music CD is paused and the audio amplifier reallocates the audio channels to enable a hands-free phone call by means of microphone and loudspeakers of the sound system.

Due to the complexity of the systems involved, integration often has to start before the components and the software are fully developed. In addition, the integrator has only limited interfaces available by means of which tests can be controlled and supervised – typically only the external interfaces of a component, such as MOST, CAN, etc. If errors occur during later integration phases it is usually difficult to assign these errors to a specific component so that a respective change order can be placed with the appropriate supplier. This chapter describes the test process from the OEMs' point of view. The following section gives an overview of the test activities of the OEMs in the context of the V-Model. Section 13.3 describes a generic test process used for all integration phases. The integration phases are explained in the following sections. They are categorized into:

- component test,
- integration test and
- system test.

Section 13.7 gives a survey of test tools.

13.2 Overview of the OEM Test Activities

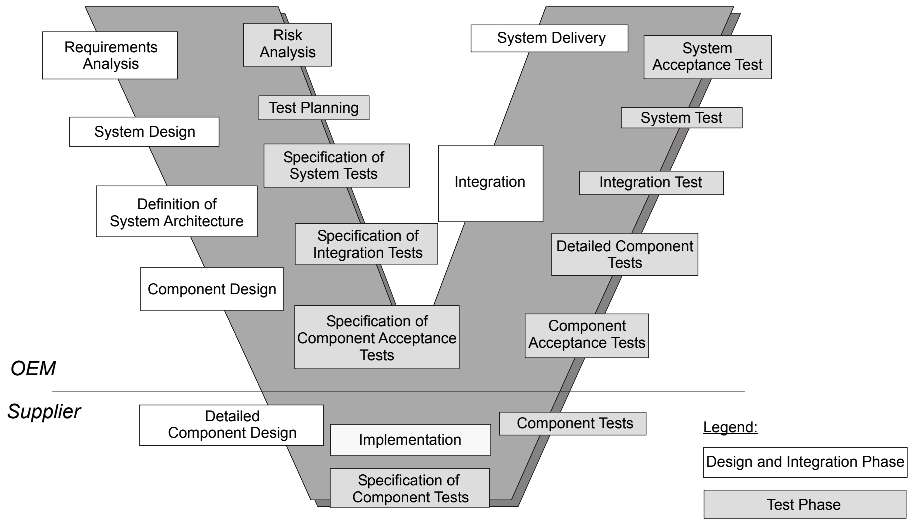

Requirements Analysis
Risk Analysis
System Delivery
System Acceptance Test
Test Planning
System Test
System Design
Specification of System Tests
Integration
Integration Test
Definition of System Architecture
Specification of Integration Tests
Detailed Component Tests
Component Design
Specification of Component Acceptance Tests
Component Acceptance Tests
OEM
Supplier
Detailed Component Design
Implementation
Component Tests
Specification of Component Tests
Legend:
Design and Integration Phase
Test Phase

Fig. 13.1: The V-Model (source: Daimler AG)

Figure 13.1 shows an interpretation of the V-Model that describes the allocation of development activities between OEMs and suppliers. The requirements on the overall system are defined first. This results in a system design, which identifies, among other things, the various components of the system. The requirements on the individual components are distributed to the suppliers in the form of a component requirements specification. The detailed design, implementation and test of each component are then carried out at the supplier. The delivered components are subsequently accepted by the OEM by means of component acceptance tests and then integrated step-by-step into the overall system. Integration tests check the correct interaction of the components. The basis for the integration tests is the interoperability specification, which is defined during the specification phase. During each test phase, the following quality characteristics are checked:

- Compliance with the functional specification
- Robustness and fault tolerance
- Performance and stability

The planning and specification of the tests can begin as soon as the requirements have been defined.

The introduction of the MOST Compliance Process (see chapter 12) has influenced the development process at the OEMs in the following manner: The MOST behavior of a component is currently specified in terms of a reference to the MOST standard and the specification of additional OEM-specific functionality. The increasing harmonization between the OEMs will result in a reduction of the OEM-specific functionality as more and more elements of the requirements specification are replaced by references to the standard or OEM defiend Test Specifications, like the Automotive Application Recommendation. The OEM can then significantly reduce the amount of in-house testing and expect fewer problems during the integration of the MOST system. As a result, the OEM can focus on confirming the OEM-specified functionality.

13.3 The Test Process

In each test phase (component test, integration test, system test) a number of activities are systematically performed.

13.3.1 Risk analysis

Risk analysis is used to identify potential risks and problems related to the MOST system under development and, if necessary, risk-minimizing measures in the architecture are defined. In addition, the risk analysis influences the planning and prioritization of the quality assurance activities during the development process. As part of which, the *test plan* defines the focus of the test activities. The test plan consists of the identification of the objects to be tested, test goals, methodology for reaching the goals, resource planning and supplier management. In the course of the following test activities, the test plan is successively refined into executable test specifications.

13.3.2 Test design

During *test design,* tests that were defined during the test planning phase are systematically refined into a number of test cases. The test design phase aims at defining a number of abstract test cases. The test cases specify the tests in such a manner that they can be understood by an application expert who is not familiar with the details of the implementation. If possible, the test cases should be reusable for numerous releases of the test object and possibly also for a number of different test objects which share common properties. By describing the tests in an abstract manner (e.g., at the same level of detail as the requirements specification), they can be reused more easily. A systematic approach should be applied when selecting test cases to ensure that the test goals are adequately covered (e.g., the classification tree method [Lehmann 2000]).

13.3.3 Test implementation

During *test implementation* executable test cases are implemented on the basis of the abstract test cases. Specific values for the test input data and the expected results are, if not already defined, selected at this point of time. If the test cases are to be carried out manually, the test case describes a sequence of actions to be performed by the tester in natural language according to a predetermined format. The sequence unequivocally defines the steps to be followed during the test. If the tests are to be carried out automatically (see section 13.7), the test cases must be programmed or generated automatically. In this context, the systems required for executing the tests must also be prepared.

The starting point of the *test execution* depends on the release of the object to be tested. For that reason it is useful to complete as many of the test preparation activities (planning, design and implementation) as possible, before the test object is released for the test. As soon as the test object becomes available, the tests can then be carried out without further delay.

13.3.4 Test evaluation

During *test evaluation* the results of the tests are evaluated and an assessment is made as to whether the goals defined in the test plan have been achieved. In the case of failed test cases or of deficiencies in the tests themselves, some or all phases of the test process may have to be repeated. The results of the test evaluation are documented as a part of the summarizing test report and issues in the test object discovered by the tests are entered into an error tracking system.

13.3.5 Test management

Test management is responsible for the correct application of the test process and for ensuring that the aims associated with the process are achieved. Test management also has to accomplish a number of other tasks. It provides a contact person for in-house (test team) and external issues (acquirer, project management, suppliers, quality assurance), it organizes and moderates reviews of test artifacts, supervises the cooperation with suppliers and is heavily involved with test planning and system acceptance activities.

13.4 Component Tests

Although component tests are already carried out by the suppliers, the OEM also performs a component acceptance test of his own. This is to ensure to the highest possible extent that the device meets the requirements of the component specification before system integration. The focus of the component test varies depending on the development status of the device.

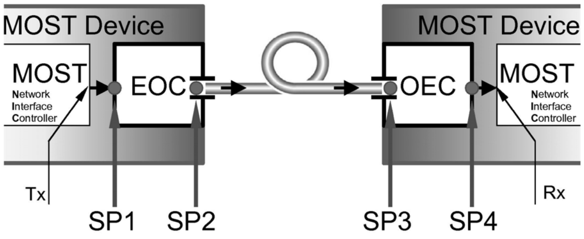


Fig. 13.2: MOST Physical Layer architecture (source: [ComPhyLP 150])

The test of the optical modules is an important aspect of this test phase. The FOT/Pigtail modules, referred to as EOC (Electrical/Optical-Converter) and OEC (Optical/Electrical-Converter) in figure 13.2, are tested within the MOST Compliance Process (see chapter 12) for conformity with the specification. The test is particularly carried out between the specification points SP1 and SP2, as well as between SP3 and SP4 (see chapter 6) [MOST 2003]. The OEM tests the correct integration of the module within the device. It is important to verify that the input signals of the FOT lie within the specified range. Typically occurring problems that have to be tested for are, for example, an incorrect peripheral circuit of the FOT, reflections if the PCB-connections between NIC and FOT are too long, or attenuations due to excessive bending of the Pigtail fiber in the device.

Further tests concern EMC interference caused by other modules (such as GSM modules) and interference of other modules in the vehicle caused by the FOT/Pigtail module itself (due to the plastic opening of the MOST connector in the device housing). In order to be able to carry out measurements of the output signal, a signal is usually generated by the NIC in normal operation and measured at SP2. In a random test it is expected that there is enough buffering between the measured signal and the predetermined tolerance limit to compensate for quality variations in large-scale production, and for ageing and temperature drift. Latter is documented within the physical Layer compliance Test.

The OEC is additionally tested at the input of the control device. A MOST conforming optical input signal is applied to SP3 and passes through the entire specified optical power range (-2 dBm to -23 dBm). A signal lock must be available across the entire range of the optical input power, in order to identify possible errors, such as increased pulse width distortion (PWD) when the amplification of the optical receiver in the intermediate ranges is switched. In order to check the timing parameters at SP4, worst case input signals are applied to SP3 by means of a test device and are measured by means of an oscilloscope.

A particular challenge in testing the MOST Physical Layer is in the test setup. Active probes with a very small input capacity are required, as well as a wideband oscilloscope. It must also be ensured during the tests that no undesired side-effects are recorded, such as line reflections due to the measuring setup. In addition, the ring architecture of a MOST System is a challenge because devices have to be measured within a closed ring.

An essential part of the component test is the testing of the MOST Network Management functionality. The tests distinguish between the component-under-test being the NetworkMaster or a NetworkSlave. The start-up of the device, the querying of device information and FBlocks as well as the shutdown are tested. It is important for further tests that the device shuts down correctly as this is the prerequisite for carrying out several automated tests successively where each test assumes that the device is in a specific state.

Additionally, functions such as ring break diagnosis, automatic repeated transmission (low, mid and high-level) and temperature management are tested. For the temperature management, the tests are carried out in a climate chamber, where exact and repeatable variations in temperature can be tested.

For component testing, especially for core compliance testing, the temperature is simulated by the FBlock ET. This method is easier and shorter than setting up the device in a climate chamber.

An important characteristic of a MOST device that is to be integrated in the overall vehicle context is its resistance to power supply variations. The static undervoltage test checks the behavior at specific constant voltage thresholds (e.g., whether a shutdown is carried out or whether the function is restricted). For dynamic undervoltage tests, the voltage is varied and the device is tested to see in which range the variations can be compensated.

The component acceptance tests are often made available to the suppliers so that they can reproduce the errors found during the acceptance tests. For that purpose the suppliers do not only need the test scripts, but also all the relevant tools, databases and documentation required to perform the tests.

Particular challenges in the testing of the network management layer are the automation of the power supply by means of controllable power supplies and the chronological accuracy of the simulations and measurements.

Due to the functional complexity of modern infotainment systems, it is increasingly important to ensure interoperability and functionality of the control devices as early as possible. It frequently happens, however, that not all control devices are simultaneously available for implementing a cross-system function. This is why the interoperability in a component test is often first tested on the basis of remaining bus simulators and test systems. The aim is to provide a system environment sufficient to enable the effectual testing of the MOST functionality of the device. Examples of functions of such a test environment are the provision of a MOST NetworkMaster and FBlock Controller to address the function blocks which are to be implemented into the device.

13.5 Integration Tests

The focus of the integration tests is to ensure the correct interoperability between the different MOST components. This integration step is often carried out under laboratory conditions on customized test boards offering easy access to all interfaces required; this enables the quick set-up and integration, and the flexible use of test and monitoring tools.

Network management functions, such as resource and connection management, diagnosis and software download can often only be tested in the integration test phase, as the functions depend on several devices that cannot be simulated without significant effort. A generic FBlock test is also carried out in this phase. This automated test queries each FBlock in the device to check whether all specified functions are implemented and behave according to the MOST Core Compliance Test Specification [Core]. For example, it is tested whether the correct error codes are sent back in the case that an incorrect OPType is addressed or the parameter limits are exceeded.

The dynamic behavior and the interoperability of the different function blocks are specified on the basis of Message Sequence Charts [Z.120 99] (see appendix A). The manual design, implementation, execution and assessment of tests of one Message Sequence Chart alone can take several days. The use of a formalized specification language, such as MSC, enables the automatic generation of tests which can reduce this effort considerably and allows a more elaborate test than before. To derive executable tests from the MSCs is, however, not as easy as it might appear at first sight and requires the following process steps.

Defining the test interface: The specification describes the cooperation of all FBlocks in the system. Depending on the integration phase, certain FBlocks or FBlock Controllers will not be available in the system for a specific test. In order to be able to execute a test fully automatically, at least one device in the system must be simulated. In integration tests, the user pressing the key, for example, is often simulated. In component tests, at least those devices communicating directly with the components to be tested have to be simulated. The test interface defines which components are to be tested and which components are to be simulated.

Modeling the test behavior: After the above described test interfaces have been identified in all MSCs to be tested, the resulting specifications are combined into one complete model. Alternative paths that are distributed over several MSCs are combined and references to other MSCs are resolved. The relationships between the individual MSCs required to build this complete model are often specified by additional *High Level Message Sequence Charts*, from which the individual test sequences can then be deduced.

Generation of test data: The messages in the MSC specifications often contain parameters which can change during runtime. A typical example is the parameter *telephone number* in a message describing the status of a telephone call. It is important when simulating such messages in a test environment to use concrete values. The selection of these values can have a great influence on the test. If, for example, the telephone number already exists in the address book, the name of the caller should be indicated. A systematic approach is required, not only to identify the relevant test sequences but also to identify with which parameter values the sequences should be executed.

Further challenges with regard to the integration testing of MOST systems are, among others, the test of non-specified system behavior (usually error cases) and the test of the many different possible system configurations (e.g., based on combinations of optional extras, country variants, etc.).

Concretization of tests: In order to execute the test cases, they have to be provided with much system-specific information not contained in the specification for reasons of abstraction (e.g., the node position address of the individual devices in the ring). A concretization step is thus necessary to generate executable test scripts from the test descriptions.

13.6 System Tests

During the system test, the MOST components are integrated and tested in the context of the final vehicle system. The focus of the system test is to test the interoperability of all MOST components and other vehicle systems (such as electrical system, CAN buses, etc.) and the stability of the individual components under "real life" conditions. It is a particular challenge to systematically cover as many situations as possible that cannot be tested under laboratory conditions. For example, unfavorable combinations of different operating conditions, which do not often occur in the field but could still lead to critical malfunctions. In this case, it is important that the causes of the error can be repeated later under controlled conditions. In this context, data logging tools are often used (see section 13.7), which can record and analyze the entire bus communication. Usually, a large amount of data is generated by such tools and must be limited by using filters and trigger conditions while taking care that the relevant data still remain intact. Another problem in carrying out system tests is the chronological synchronization of the different bus systems so that the reaction of the MOST bus to messages from other buses (e.g., the serial connection to the GSM phone) can be reproduced. In order to identify causes of errors it may be necessary to also influence the MOST physical layer in the vehicle (e.g., to reduce signal power in a particular section of the MOST ring).

13.7 Test Tools

Test environments and test tools are necessary for different test activities to ensure a systematic, efficient and repeatable test. Standard, off-the-shelf test tools only support parts of the entire test process. For this reason, a tool suite is required, consisting of a combination of commercial and proprietary tools and adapted to the individual requirements of the OEM-specific processes. When using test environments and test tools, it must be ensured that they are technically mature and that their use does not lead to error reports which are not related to the actual test object. This section gives a survey of the test tools relevant for the test of MOST systems.

Test Management Tools such as TestDirector [Mercury] support test planning and test traceability. Requirements can be imported and referenced during the definition of test cases. In this way, test results can be directly assigned to the relevant requirements. Test management tools often also include defect management functionality. These functions support the management of defects that were found during the tests and the tracking of the elimination of these defects. Test management tools also provide functions for summarizing the results of the test process (number of tests carried out, number of defects not yet eliminated, etc.) in the form of charts or reports so that they can be used in reporting the quality of the system to project management.

Tools for the automation of test execution activities can be roughly divided into two groups: Tools that are responsible for monitoring and recording the MOST behavior on the MOST bus (Tracing Tools) and tools that send specific stimuli to the test object on the basis of predefined scripts and compare the reaction to the expected MOST signals. Both types are often used together to obtain as complete a picture as possible of the behavior under test.

Important characteristics of MOST Tracing Tools are, amongst other things, the chronological exactness of the traces, the ability to decode recorded messages by means of the XML-based function catalog specification (in real time), the automatic composition of segmented messages, the monitoring of Packet Data and Control Channel, the definition of complex trigger and filter conditions, as well as the access to the Streaming and Packet Data Channel. Some tracing tools can additionally be used as data loggers for installation in the vehicle even without a connection to the PC or laptop. Such tools are particularly well suited for system tests, as they can easily be built into the car and can continuously log the MOST Bus. Tracing Tools for MOST systems have been developed by SMSC [SMSC], Vector [Vector] and Optitas [Optitas].

Test script-based Test Automation Tools for MOST are offered by GADV [GADV], Ruetz Technologies [Ruetz], and Vector Informatik [Vector]. The tools do not only offer a MOST interface, but also the support of several vehicle-specific bus systems at the same time. Important characteristics of such tools are, among other things, the clear evaluation of the test results, the chronological exactness of the system when the results are measured, and the ability to dynamically react to incoming messages.

The *Testing and Test Control Notation* (TTCN) is a standardized test technology. The latest version TTCN-3 [ETSI 2003] was standardized by ETSI (ES 201 873 Series) and ITU-T (Z.140 Series). TTCN-3 is a very flexible, abstract and expressive test specification language, which also has a graphical presentation format strongly based on MSC-2000. This allows the description of integration tests at the same level of abstraction and in a similarly readable form as the specification. Therefore, TTCN-3 can be a suitable means for specifying standardized tests in the future – in line with its origins in the standardization of Compliance Tests for telecommunication protocols. TTCN-3 also standardizes, besides the test language, the operational semantics of the language so that a uniform behavior of the test scripts in different tools can be guaranteed. This would allow the exchange of MOST tests between OEMs and suppliers without the use of a specific test tool having to be specified. Some OEMs [Burton 2004] and MOST Test Tool manufacturers [Ruetz] have already examined TTCN-3 for its ability to meet the challenges with regard to the testing of MOST systems.

14 Introduction of MOST150 in Series Production

14.1 Application of MOST150

Various vehicle projects have used MOST25 technology; it is characterized by a high quality index and a strict focus on the requirements of automotive infotainment systems. Over the years, expert knowledge was acquired along the process chain (development, quality assurance, production, and customer service) and the processes were established. With requirements towards car-based infotainment systems becoming more challenging, for example, video transmission and Ethernet connectivity, MOST25 maxes out, in particular in terms of available bandwidth (see fig. 14.1). Consequently, the path was cleared for the third generation of network technology: MOST150.

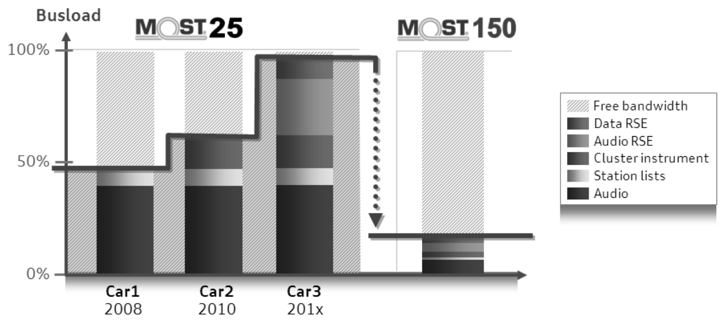


Fig. 14.1: Bus load increase in Audi systems

As the logical continuation of existing MOST25 transmission technology, it offers the following distinct advantages:

- Six-fold bandwidth increase compared to MOST25 (see fig. 14.1)
- Native Ethernet channel for transmission of IP data.
  This allows flexible introduction of IP applications.
- Isochronous streaming
  Can be used to transmit video streams (MPEG transport streams, VideoOver-MOST)

- Reuse of MOST25 applications
  MOST25 applications, which are in use in current infotainment systems, can be adopted almost without modifications

MOST150 multi-channel network

MOST150 enables the parallel use of all the mentioned services (packet data transfer and time-synchronous streaming) within one bus system. Depending on the system design, the individual services can be dynamically assigned with different resources during run-time.

14.2 MOST150 technology evaluation

14.2.1 Requirements and evaluation structure

The precondition for the introduction of MOST150 is a level of maturity comparable to MOST25, that is, the proof of suitability for series production. Every OEM will thoroughly test new technology before deploying it. If the subject is - as is the case with MOST150 - the result of an evolution of an existing system, attention is paid to the continuing use of present qualifications, systems, and processes. The main focus, however, remains on improvements and new functions. Only when the critical requirements are met and the quality of software, hardware, tools, and testing systems clear the path to series production, a switch in technology is feasible.

Figure 14.2 presents the layers that are affected by the migration from a MOST25 environment to MOST150. The changes range from the physical layer to the application layer. This creates a challenge in terms of structuring the evaluation project so that a verification of all relevant elements becomes possible.

In the case of MOST150, the evaluation can be structured to include the following work packages:

- **Physical Layer**
  This includes the verification of all relevant hardware components (POF, FOT, Pigtail, INIC) and the corresponding electrical parameters.
- **Network Services/INIC firmware**
  These form the software interface to the MOST bus and have significant influence on performance and reliability.
- **Network management/power management**
  These functions, which are elementary to every network, are required for administration, wake-up and sleep behavior.

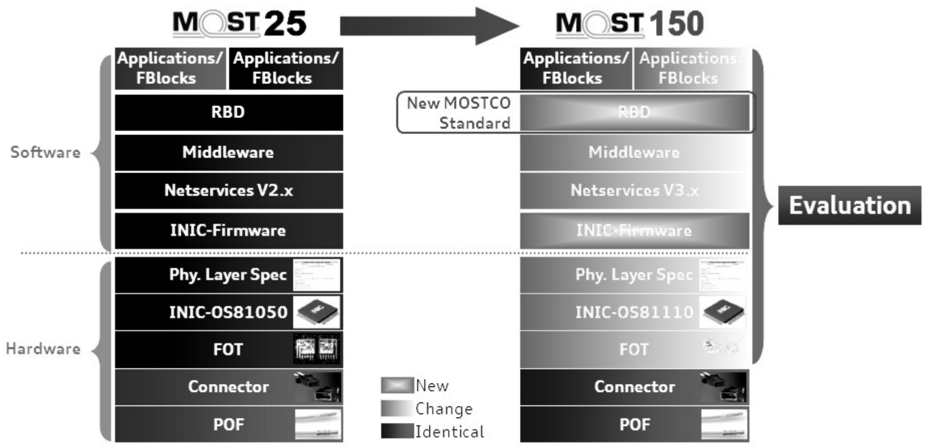


Fig. 14.2: Layers that are affected by the migration

- **MOST diagnosis**
  Verification of MOST diagnosis covers ring break diagnosis (RBD), Electrical Control Line (ECL) which was first standardized with MOST150, and the new diagnosis function "Sudden Signal Off / Critical Unlock" (SSO/CU) enables detection of the originators of sporadic ring breaks and Critical Unlocks.

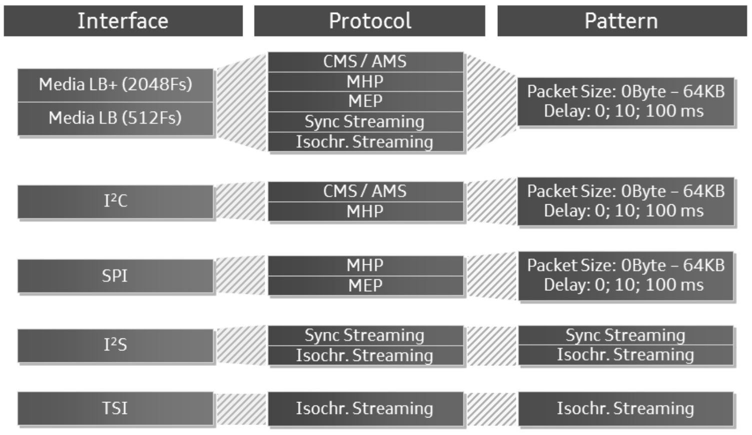


Fig. 14.3: Definition of use cases for the evaluation

- **Use Cases**

MOST150 will be deployed in different configurations and electronic control units. To cover all MOST150 applications that are relevant to OEMs, use cases are generated through abstract modeling. Based on actual application scenarios (e.g., transmitting a title list from a media player to the head unit), the departments for networking and infotainment define combinations of INIC interfaces, network protocols, and the corresponding data patterns (see fig. 14.3). The collection of use cases makes it possible to verify all INIC interfaces (MediaLB 3-pin, MediaLB 6-pin, I^2C, SPI, I^2S, and TSI), MOST protocols (CMS/AMS, MHP and MEP), and MOST channels (Control Channel, Packet Data Channel, Streaming Channel).

- **Tools**
  These are indispensible for series development of MOST150 components. Active tools and listen-only tools, as well as the integration into the test automation have to be verified.

As the preliminary consideration shows, a variety of requirements for the evaluation of technology regarding maturity for series production has to be taken into account. Subsequently, the example of practical implementation in the Volkswagen group highlights one possible way of evaluation.

14.2.2 MOST150 reference platform

The multitude of requirements towards the technology evaluation demanded the establishment of two verification projects. The hardware of the first project, the so-called reference platform, consisted of a MOST ring with seven participants (evaluation boards), which included TimingMaster, NetworkMaster, and PowerMaster. The focus of this system was the verification under laboratory conditions of all work packages that had been identified. The design of each evaluation board allowed, for example, switching MOST hardware through the use of easily exchangeable pluggable circuit boards (“daughter boards”) with INIC and FOT. Initially, the emphasis was on tests of the wake-up/sleep behavior, reliability, and bus diagnosis (RBD, SSO/CU). Furthermore, the system provided access to all INIC interfaces so that the 72 previously defined use cases were completely executed on this platform. Hereby, all relevant application scenarios for Audi and the Volkswagen group could be verified. The third pillar was formed by thorough performance measurements of all MOST channels and transmission protocols.

Due to its layout, the MOST150 reference platform is not suitable for vehicle deployment.

14.2.3 MOST150_InCar

The second system was used for prototypic MOST150 integration in the vehicle. Of particular interest in this scenario was the influence of the car environment on MOST150 communication.

Tests were conducted in the following areas:

- overvoltage, undervoltage - variation of supply voltage by the on-board electrical system simulator.
- temperature stress - passing through different temperature profiles in the climate chamber.
- wake-up/sleep behavior - automated waking of the vehicle's buses in varying time intervals.
- test drives - verification of communication in a customer environment (check of received patters for absence of errors)

Apart from these tests, MOST150_InCar included a reference implementation, which served the confirmation of interoperability of different ECL transceivers and the verification of ring break diagnosis.

14.2.4 Results of the technology evaluation

After performing and completing the evaluation, the following results - presented in the same order as the evaluation structure - were obtained:

- **Physical Layer**
  Verification of all relevant hardware for MOST communication was successful. Parts availability for series development of components, which was investigated in parallel, was confirmed, as well.

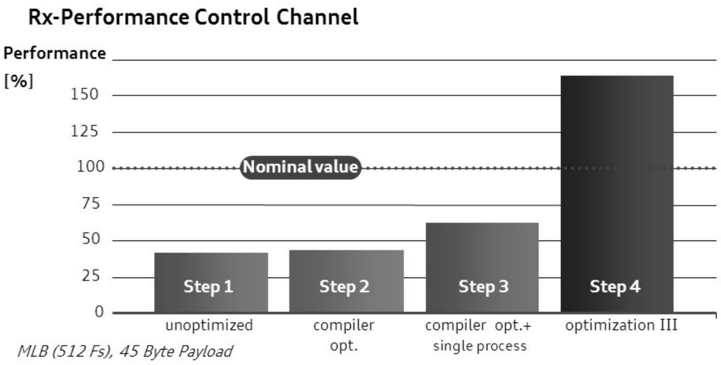


Fig. 14.4: Control Channel optimization steps

- **Network Services/INIC firmware**
  Different Network Services and INIC firmware versions were verified in terms of performance, function, and reliability. Detected inconsistencies were resolved, which led to significant performance improvements in critical areas.

Figure 14.4 exemplarily shows the throughput on the Control Channel. Here, the implementation and trial of different optimization measures were performed; eventually through the implementation of the so-called Network Services "Optimization III", the previous internally defined reference value was even exceeded.

- **Network management/power management**
  To verify this functional area, several ten thousand wake-up/sleep cycles were invoked. Behavior was tested under conditions of network stress, voltage stress, and temperature stress. On this broad statistical basis, a positive verification of this area could be achieved.
- **MOST diagnosis**
  Besides verifying ring break diagnosis, evaluation of the new features SSO/CU and ECL communication rendered positive results. Hence, apart from the diagnosis of static interruptions, which was already defined for MOST25, now detection of short breaks and communication in the case of an error is uniformly specified and available across OEMs.
- **Use Cases**
  The core of the technology evaluation is the testing of the relevant use cases. These tests were conducted using different patterns (variation of packet size and timing), where each pattern was applied for at least 6 hours. Based on more than 4,800 hours of test execution, the successful evaluation of the application scenarios was performed. As a byproduct, by means of these tests, the process chain from supplier over implementer to tester and developer was checked for weaknesses and optimized. Thus, countermeasures for detected errors could be provided faster through the corresponding communication paths, integrated, and subsequently re-tested. Such processes are of great importance for minimal-risk development.
- **Tools**
  The measurements that were undertaken in the scope of the verification projects used tools that due to the novelty of MOST150 had to be newly developed. In a number of cases this led to the problem that malfunctions had to be assigned to the technology itself on the one hand or the implementation in the tools or the measuring software on the other hand. A close collaboration with the manufacturers was required so that remedy could be provided for the errors that occurred in that domain.

14.3 Conclusion

The use of two platforms (MOST150 reference platform and MOST150_InCar) allowed to verify all elements that were affected by the migration from MOST25 to MOST150, both under laboratory conditions and in road tests.

All deviations of the base technology that were detected in the course of the evaluation could be corrected. Expectations regarding data rates, reliability, functions, as well as availability and quality of parts, were met. Based on the findings resulting from this evaluation, the transition of MOST150 into series production was made. There still remains potential for optimization regarding performance and processor load, especially for applications that utilize the MOST High Protocol. These optimizations can be introduced into series projects by later updates of MOST Cooperation specifications and supplier software.

The gathering of expert knowledge that is connected to the evaluation project is a huge benefit for further series development. Quality is boosted through the adaptation of the Network Services integration process, which was extended as follows:

After the release of new Network Services or INIC firmware versions, preliminary integration is performed, followed by regression testing on the MOST150 reference platform. Only when the tests have executed successfully, integration into the infotainment devices takes place.

This guarantees interoperability of different versions and specific configurations on MOST system level.

14.4 Outlook

MOST150 provides—through bandwidth reserve and functions for connecting consumer electronic devices—a future proof infrastructure for infotainment networking.

Due to the technological advantages, MOST technology could in the future become established in additional vehicle applications. A potential area is the field of driver assistance systems. The requirements of image processing and sensor data processing towards networking are showing parallels to the requirements in the infotainment domain:

- The complexity of image processing systems and infotainment system is comparable.
- The transported data types (packet data and streaming data) exhibit great similarity.
- Audio data and video data have to be transmitted time-synchronous and with minimal delay.

Further concept analyses can bring clarity to which extent MOST can meet the requirements towards topology and reliability of transmission.

15 MOST150 Migration

15.1 Introduction

This chapter presents two possibilities for migration from MOST25 to MOST150. One of the proposed solutions for the migration is an evolution concept, where all MOST25 devices are upgraded to MOST 150, using a special hardware interface. The second proposed solution is the development of a specific bridge architecture between MOST 25 and MOST 150. The MOST Cooperation favors the evolution approach.

15.2 MOST150 Evolution Approach

When migrating from MOST25 to a MOST150 infotainment system directly, this approach is called the *MOST150 Evolution Concept* [Schr 01] [Schr 02]. The basic of that concept is the idea to upgrade all devices to MOST150 off the shelf by adopting the hardware and the system software of the respective ECUs. The advantage with regard to the wiring harness costs is that no changes are necessary to the established wiring harness of a MOST25 system, compared to the *MOST25/MOST150 Bridge Concept*.

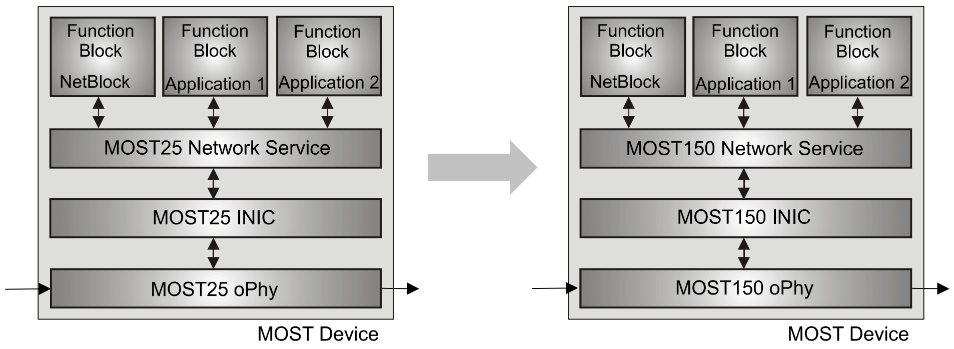


Fig. 15.1: Illustration of the MOST150 Evolution concept

15.2.1 Hardware Evolution

The aim of the Evolution Concept is to point out that it is possible to migrate an ECU “directly” without the need for a redesign of the complete ECU’s hardware. However, the precondition is that the respective MOST25 device, to be adapted to MOST150, uses an INIC architecture. If so, the principle is similar to exchanging the network card of a common PC as presented in figure 15.1.

The basic workflow when applying the concept is to remove the network interface card, and plug in and update the drivers for the new one. To speak in automotive ‘MOST language’ that means to replace the MOST25 components, INIC and FOT, by the new MOST150 components, compile the ECU software with the MOST150 Network Services and finally flash that firmware on the device.

However, unlike the consumer electronics world – using PCI network interface cards for example – there is no compatibility between the MOST25 and MOST150 components with regard to the physical connector pinning. Hence, in order to realise the *MOST150 Evolution Concept*, a kind of MOST150 network interface card, called *MOST150 PhyBoard, is required*:

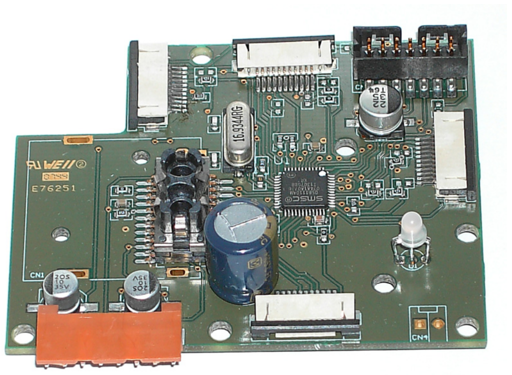


Fig 15.2: MOST150 PhyBoard (source BMW AG)

The PhyBoard includes the MOST150 FOT as well as the MOST150 INIC and all necessary peripheral components, such as resistors, capacitors, crystal and so on. The ports of the new MOST150 INIC (OS81110) are wired to 4 FFC (Flat Flexible Cable) connectors. The pin assignment of these connectors was chosen by the author in a way that corresponds to the pinning of the MOST25 INIC (OS81050). On this account, the PhyBoard can be connected one-to-one to the footprint of the MOST25 INIC on the circuit board by the use of FFCs (Flexible Flat Cables). Hence, the adaptation of an existing ECU is achieved as presented in figure 15.3.

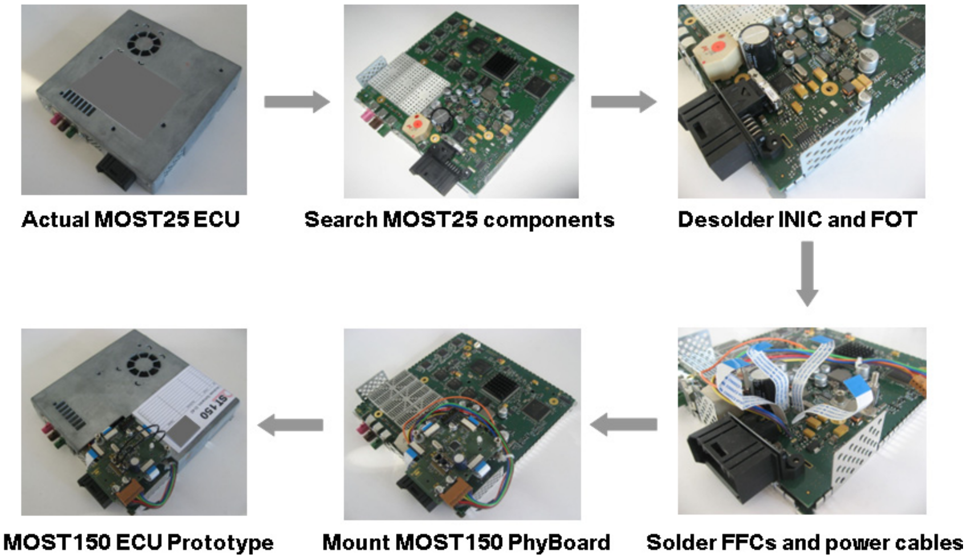


Fig. 15.3: MOST150 Evolution Concept workflow (source BMW AG)

The ECU to be adapted is opened and the MOST25 INIC, as well as the MOST25 FOT, are de-soldered from the PCB. Afterwards, the FFCs are soldered onto the footprint of the INIC. Power for the PhyBoard is taken from the original PCB by soldering cables to the footprint of the FOT. Next, the MOST150 PhyBoard can be mounted on the ECU's PCB quite simply by the use of spacing rollers. Afterwards, FFCs and the power cable are plugged in. In doing so, one gets a fully MOST150 enabled ECU prototype hardware. The only thing to be done next is to upgrade the software with the new Network Services and flash the adapted ECU.

Figure 15.4 illustrates this concept. Several ECUs have been adapted to MOST150.

15.2.2 System Software Evolution

The FBlocks for NetworkMaster, NetBlock, ConnectionMaster, as well as relevant functions from the GeneralFBlock template only have to be updated. The applications themselves stay untouched. However, a minimum MOST 150 feature set has to be implemented. Following, a few examples of mandatory features are provided:

- The system has to support the new frame structure.
- Functions from the FBlock bundle 3.0 replace functions in the current system. The FBlock bundle 3.0 contains the GeneralFBlock, NetBlock, NetworkMaster, and the ConnectionMaster.

- The max. AMS telegram payload size was increased from 12 to 45 bytes. Consequently, the TelLen field becomes 12 Bits large, and now has the same size as the TelLen field in the MOST High Protocol.
- The sample frequency no longer is in a range between 30 kHz and 50 kHz but either 44.1 kHz or 48 kHz. 48 kHz is recommended.
- "Mandatory" and "Extensions" ranges of functions are now combined in the "Application" range.
- NetworkSlaves now respond with an unchanged list if the InstID cannot be changed.

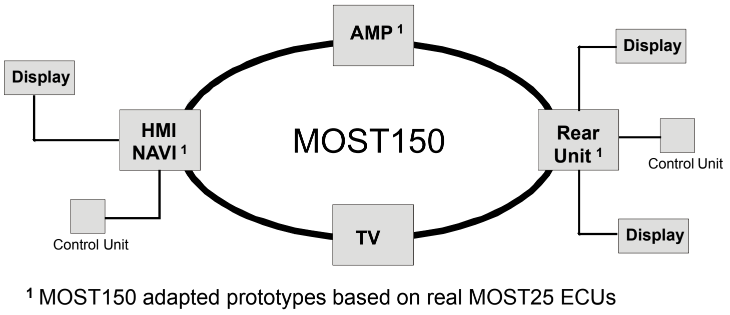


Fig. 15.4: Principal MOST150 Evolution Concept set-up

- Configuration.Status now uses Control=NewExt, which also provides the DeviceID.
- Function classes were modified (Container, restrictions on the use of streams, stream signals…).
- If the Central Registry is "full", that is, no new FBlocks can be registered, the NetworkMaster replies with Configuration.Status(NewExt, <empty list>). If some but not all FBlocks can be registered, NetworkMaster.Configuration.Status(NewExt, <partial list>) is sent.

Beyond that, all implementations of MOST25/50 concepts have to be removed that were abandoned for MOST150. These are, for example:

- MAMAC is no longer available. If MAMAC is used, it is replaced by Ethernet over MOST in MOST150.
- The SourceConnect/SourceDisconnect approach, which is specific to MOST25, has been abandoned.
- Secondary Nodes are not supported.

- MOST High over Control Channel is not supported.
- The “data link layer 48 bytes mode” is no longer required; however, the maximum payload may still be limited by device specific characteristics (e.g., used I/O interface or buffer sizes).
- The SourceHandles function is no longer supported.
- Large updates to the Central Registry are no longer announced by sending NetworkMaster.Configuration.Status(OK).

Additionally conditional and optional features have to be considered. Conditional means, for example, if the system does not use the Packet Data Channel, there is no need to support low level retries on that channel. Optional features are new MOST150 features, for example, isochronous transfer was introduced, supporting the use cases DiscreteFrame Isochronous, A/V Packetized Isochronous, and QoS IP.

Note: If you have to migrate to MOST150, attend to the latest revision of the *Application Note MOST150 Migration* from the MOST Cooperation [AppN Mig].

15.3 MOST25/MOST150 Bridge Concept

The second proposal, when migrating from the actual MOST25 infotainment system to a MOST150 based one, is the so called *MOST25/MOST150 Bridge Concept*. That approach assumes that at first only the components with high bandwidth requirements are upgraded to MOST150. The other nodes continue to be MOST25 based. Obviously, two optical rings are required with this concept, linked by an additional new device, a so-called *MOST25/MOST150 Bridge*. Although the bridge could be integrated in an ECU, e.g., the HU, this concept is cost-intensive and, hence, is not recommended.

15.3.1 General Bridge Architecture

The bridge device is intended to be used to connect a MOST25 network to a MOST150 network, whereby streaming, packet and control data need to be routed. Figure 15.5 gives an overview of the general bridge architecture.

Both MOST INICs (INIC_A and INIC_B) use the respective MediaLB interfaces [MediaLB] to connect to the EHC (Enhanced Host Controller). MediaLB (Media Local Bus), or MLB, is a MOST specific high speed interface, providing full access to the whole bandwidth of the MOST bus. For synchronous data bridging, the

INICs are linked via their I2S streaming ports, in order to transport the data content of synchronous channels directly without the need of interaction of the EHC.

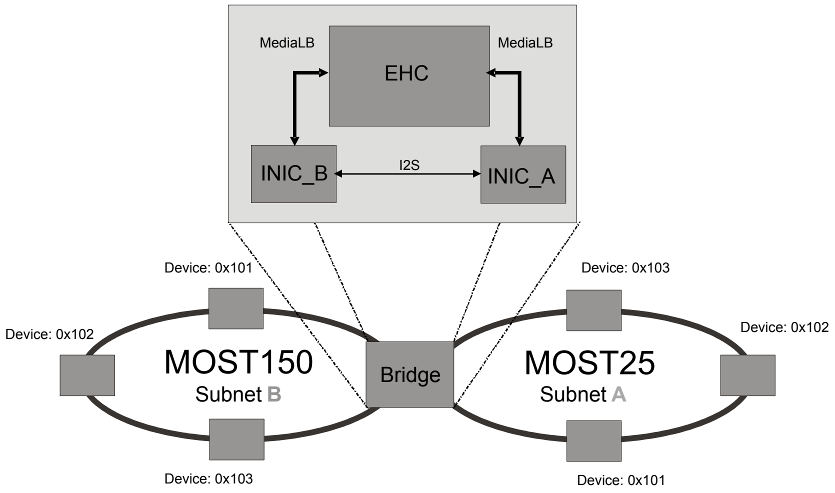


Fig. 15.5: General architecture of a MOST25/MOST150 bridge device

15.3.2 Addressing Concept

For seamless migration, it is essential that a common MOST device can be integrated into a subnet without change. Thus, the usual addressing method needs to remain unaffected by the bridge and the local addressing inside a subnet is not allowed to differ from the addressing found in today's MOST networks.

A method for addressing a target beyond the bridge in another subnet had to be defined. A solution is quite obvious. As common MOST devices, so far, only make use of the three lowest nibbles (12bit) of the device ID, and ignore the four highest ones, the idea is that the highest nibble (4bit), henceforth called *SubnetID*, of MOST's DeviceID should be used for subnet addressing. The message sequence chart in figure 15.6 depicts how the source address and the target address are mapped through the MOST25/MOST150 Bridge.

If a message, incoming from a subnet, is passed to the EHC of the bridge, the device addresses, respective *SubnetIDs*, need to be adapted before forwarding to the opposite subnet as follows: Device 0x0101 of subnet A in figure 15.6 wants to send a request to device 0x0102 in subnet B. As the target device is located in the opposite subnet, it is registered by the bridge's special NetworkMaster with DeviceID 0xB102 in subnet A. So device 0x0101 addresses its request to that ID. The bridge processes that telegram by adapting source and target device addresses before forwarding to subnet B. *SubnetID* A will be added to the source address and at the same time, the highest nibble of the target address will be set to zero as it needs to be in the local subnet. The response of device 0x102 of subnet B to device 0x0101 in subnet A is processed in the same manner.

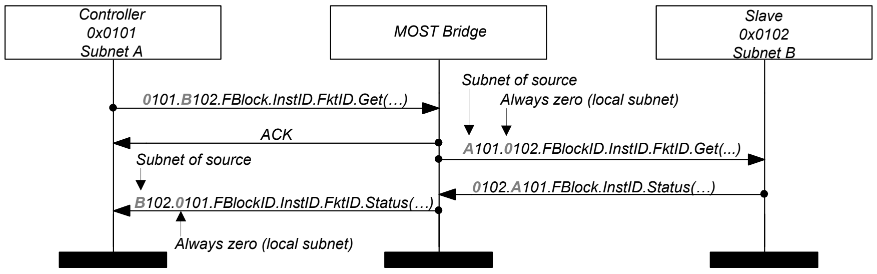

Controller
0x0101
Subnet A
MOST Bridge
Slave
0x0102
Subnet B
Subnet of source
Always zero (local subnet)
0101.B102.FBlock.InstID.FktID.Get(...)
ACK
A101.0102.FBlockID.InstID.FktID.Get(...)
Subnet of source
0102.A101.FBlock.InstID.Status(...)
B102.0101.FBlockID.InstID.FktID.Status(...)
Always zero (local subnet)

Fig. 15.6: Message sequence chart description of the subnet addressing method

15.3.3 Network Management

It is essential that each subnet is managed by its own NetworkMaster. Therefore, a network scan has to be done only within the local network. The configuration states of all subnets are completely independent from each other. A change in the configuration state thereby is recognized in other subnets by broadcasting added or removed services.

The central registry of each subnet contains all MOST services found in the whole system. Services in the local network are stored with their 'normal' DeviceID (high nibble equal to 0). Services located in the remote subnet are stored with a DeviceID in which the high nibble represents the subnet identifier of the appropriate network. Therefore a device which requests the central registry always has the ability to communicate with the correct services without the necessity to use additional mechanisms.

Figure 15.7 presents a proposal for a bridge from a network management point of view. Here the bridge acts as NetworkMaster in both subnets. A constraint to this concept is that, if the bridge is not integrated into the head unit – as the head unit recently normally is the master in a legacy MOST25 system – it will be necessary to implement the head unit as a slave device.

However, the advantages of that 'master-bridge' concept are:

The validation of MOST service instances (FBlockID / InstID) can be done across all subnets. Therefore, it is possible to solve all address conflicts inside the bridge.

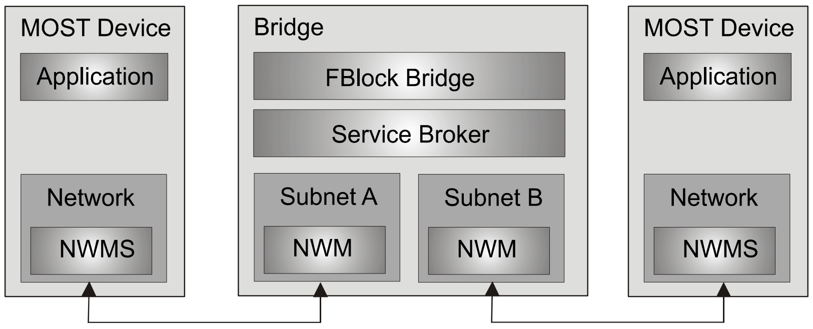


Fig. 15.7: Network management bridge architecture
NWMS: NetworkMaster Shadow
NMW: NetworkMaster

If exactly two subnets are coupled by a MOST bridge it is an advantage to implement both NetworkMasters inside the bridge. In this configuration a common instance may validate the registries of both subnets. So the pairs of FBlockID and InstID can be made unique over both subnets.

The 'master-bridge' contains the TimingMaster for both subnets to ensure that the system frequency is 100% identical. This allows the connection of the two INICs directly via the I2S interface to transmit synchronous data between the subnets without the need for buffering or sample rate conversion (refer to chapter 15.3.4).

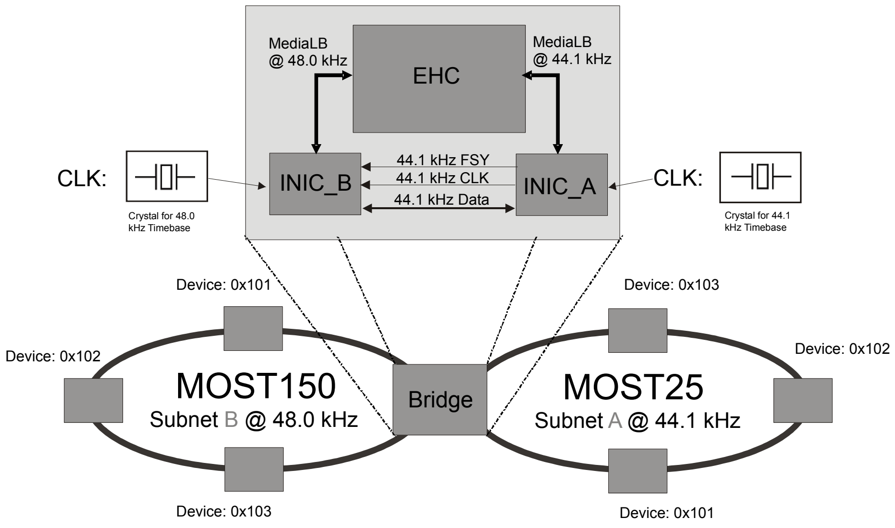


Fig. 15.8: Bridge architecture for bridging asynchronous sub-networks

15.3.4 PCM Data Bridging

If the system frequencies differ, either a sample rate conversion has to be implemented or isochronous transmission in the MOST150 network has to be used for the handling of MOST25 synchronous data. Figure 15.8 illustrates a legacy MOST25 ring with 44.1 kHz and a MOST150 ring with 48 kHz. As MOST150 provides isochronous capabilities, synchronous/isochronous data also can be linked using the streaming ports.

In addition, the establishing of the synchronous connections in both networks via the bridge has to be considered.

For more information about the bridge concept, attend to the latest revision of the Application Note MOST150 Migration from the MOST Cooperation [AppN Mig].

16 Manufacturing and Processing of MOST Components

The manufacturing process of devices with an optical MOST interface involves a number of changes and amendments compared to the manufacturing process of electronic components without opto-electronic interfaces. These changes and amendments can be categorized into two groups.

On the one hand there are parameters which are influenced by the processing of the opto-electronic components and their characteristics and which require particular measures compared to conventional electronics components.

On the other hand completely new processes and methods must be established for manufacturing the opto-electronic MOST components, which are not used normally in the manufacturing of electronic components.

Both aspects will be discussed in the following sections and the requirements with regard to the manufacturing process will be described in detail.

16.1 Structure and Testing of Opto-Electronic MOST Components

The assembly of electro-optical (FOX, Fiber-Optic Transmitter) and opto-electrical (FOR, Fiber-Optic Receiver) converters, mechanical adaptations and, as the case may be, fibers inside the device is represented by a pigtail. It depends on the respectively realized form of the product, whether the *pigtail* module must be treated as a mechanical unit in the manufacturing process.

There are two basically different mechanical design concepts for pigtails. One comprises the group of flexible pigtails with a horizontal or a vertical fiber output. They are connected to a connector by means of an external pair of fibers (here referred to as “set of fibers”). The connector is the optical MOST interface to the wire harness. The distance between the fiber face end and optically active elements (PIN diode or optical transmitter) must be kept as small as possible at both ends of the fiber.

The other structural concept comprises the rigid pigtails (with the product names µPigtail or Minipigtail, for example), where no external set of fibers is used. The FOX and FOR are integrated into a connector into which the optical plug-in connections at the side of the harness can be directly plugged.

In this case, the connection between FOX or FOR and the plug-in connection of the harness is effected via short POF pieces, glass sticks or via lens structures for optically bridging the gaps.

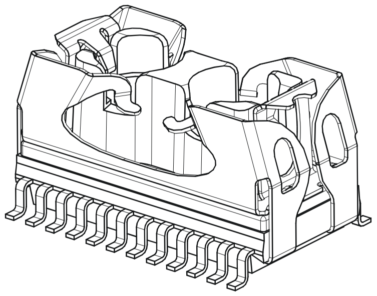


Fig. 16.1: SMD FOT as complete pigtail header solution as specified for MOST150

16.1.1 Header

In this context, a header is meant to be the mechanical entity comprising the following functional elements: optic-electrical conversion, lightguide alignment, lightguide fixture, EMI- and crosstalk shielding. The header may be an assembly of individual parts each providing one or more of these aforementioned functions. For header products using FOTs in Through Hole Mount (THM) package technology these sub-parts are typically the FOX and FOR, a required plastic adapter, the respective sheet-metal case and a holding fixture for the fibers.

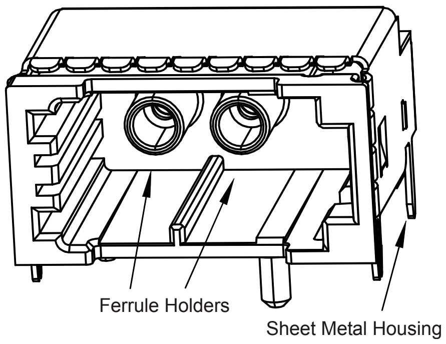


Fig. 16.2: Rigid Pigtail without additional electrical contacts

In case of a rigid pigtail header, additionally there are optical interface elements bridging the distance between the FOX/FOR and the plug-in connections of the harness. In exchange, the fixtures for the internal fiber set are dropped. Figure 16.1 shows a header in SMD package technology based on SOIC24 standard. For MOST150, this SMD-package is specified in the MOST150 oPHY Sub-Specification. As it includes all interfacing and holding elements for the device internal fiber set, as defined above, it is to be assigned to the class of headers.

In contrast to this, the ferrule adapters in the rigid pigtails serve for a direct contact to the vehicle wire harness. Header and connector form one unit. Figure 16.2 shows an example of the constructive design of a header in the rigid version. In this version there is no connection for an internal fiber set.

Functional Elements of a Header

Within the class of headers different technical solutions can be found on product level. Despite their actual technical realization they have to provide a set of functions. These functions can be realized by dedicated elements. Several functions can also be combined into sub-components.

The main function of a header is to convert the data signal from the electrical into the optical domain or vice versa. Therefore, opto-electronic semiconductors are bonded onto a leadframe or pre-molded leadframe. The leadframe, with attached IC-dies, gets packaged by insert molding, using transparent mold compound or by opaque while relieving appropriate windows for the optical signal. For pre-molded leadframes, a transparent globetop may be applied.

For FOR and FOX packaged separately, a dedicated component is used to mechanically support the FOTs and to fix them at a specific relative position. Some spacer designs additionally take special care to restrict the position tolerance of the THM-pins of the FOTs.

A mechanical interface structure, giving reference to the optical axis, is needed to enable sufficient alignment of the pigtail fiber or lightguide element. To ensure good optical coupling—under all conditions and over lifetime—the fiber ferrule must be fixed to the alignment structure. This fixture can be designed for repeatable attaching and deattaching of the fiber or for one time attachment. With the SMD package, a clamp fixation is specified that holds the ferrule into its position when a specific insertion force is exceeded. The pigtail fiber can be detached from the header by pulling with a force exceeding the retention force.

An appropriate shielding may be required to guarantee EMI conformity. A shield case could additionally prevent electrical and optical crosstalk between receiver and transmitter unit. In many header designs, the shield case provides the overall fixation and stabilization of the assembly.

Structure

By means of a suitable assembly process, the components are assembled in the sheet metal case in such a manner that the FOXs or FORs in their target position are fixed relative to the fibers or the optical path.

The actual alignment of the optical axes is not effected by means of the sheet metal case or the plastic adapter, as their tolerances are too high. It is rather the cavity of the FOX or FOR case itself which is used for alignment. In all constructions, the optical waveguide extends into the cavity of the FOX or FOR. Thus it is possible to determine the relative position towards each other of the converter and the fiber face. The position tolerances only depend on the tolerances of the cavity and the waveguide.

Testing

After the mechanical assembly and before the actual mounting, the headers are subjected to a performance and function test. In the test, system-relevant parameters (radiation power (FOX), timing parameters at low input power (FOR) and power consumption) are measured and documented. This ensures that only good parts are mounted into a device.

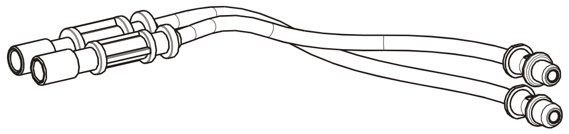


Fig. 16.3: Pair of fibers for the internal connection of the MOST150 SMD header to the device connector

16.1.2 Set of Fibers

Depending on the design (flexible pigtail or rigid pigtail), respective sets of fibers are used within a device. In contrast to the wire-set fibers, the internally used fibers have a single coating and are thus considerably thinner (ca. 1.5 mm) than the harness fibers. A set of fibers consists of two POF fibers of a predetermined length which have ferrules attached to their two ends. The ferrule geometry at the wire harness end must comply with the MOST Specification- In contrast to MOST25, the internal ferrules on the MOST150 SMD-header side are specified within the MOST Standard, too. Thus, the fiberset and SMD-header are compatible independently form their respective manufacturer. For MOST25, the ferrule at the side of the header can be designed differently depending on the manufacturer of the FOX or FOR. Figure 16.3 shows such an internal set of fibers. On the one side one can see the ferrules with the Standard MOST Interface, on the other side the ferrules with the proprietary internal interface.

The internal ferrules are considerably smaller than the ferrules with the MOST geometry. The length of the POF depends on the respective requirements of the device.

POF

The POF is cut into pieces of required length by means of a saw. The basic structure of such a separating device can be found in figure 16.4. Arrows in the sketch show the direction of transport of the fiber.

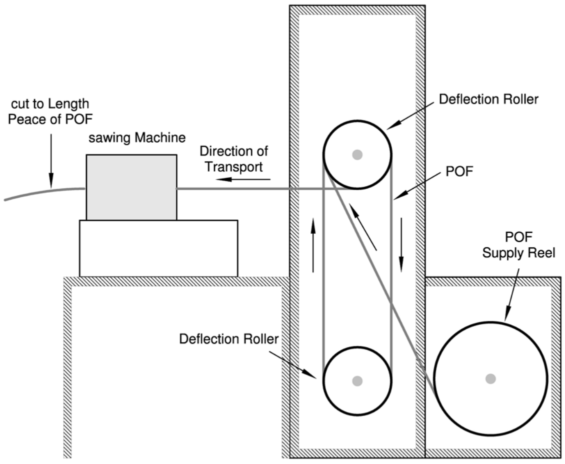


Fig. 16.4 Structure of a separating device for POF fibers

The POF raw material is delivered on a roll. The POF feed roll feeds the saw unit. The fiber is deflected by a number of deflection pulleys to ensure constant tension and avoid mechanical stress on the fiber. This reliably prevents the POF from being damaged by the unwinder-sawing fixture.

It is, moreover, very important that the end faces of the fibers not be damaged or soiled by the sawing process. Thus specific saw blades developed for that application are used in the sawing fixture. The blades have to be cleaned at regular intervals.

Besides the optical characteristics, the temperature stability is an important parameter for specifying the POF. The POF currently used in the wire harnesses of vehicles is specified for temperatures of up to 85 °C. In contrast to this, the fibers used inside the devices are approved for temperatures of up to 105 °C. Excess temperatures during processing which exceed the limiting values can cause deterioration of the optical performance and are thus to be avoided by all means.

Ferrules

There are several different basic methods of attaching ferrules to an optical wave guide. These methods comprise gluing, crimping or laser welding. The method used determines the manufacturing process of the ferrules and the costs involved. For standard MOST products, the laser welding process is used for connecting ferrules and POF. With a high power short-pulsed laser both the black plastic coating of the fiber and the surrounding ferrule material are melted. When they harden again, the ferrule and the POF coat form a high-strength plastic connection. The cross-sectional drawing of such a welded connection is shown in figure 16.5 (dashed oval line). The actual welded joint is only recognizable due to the missing air gap.

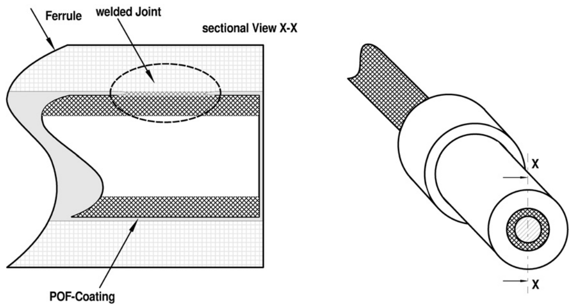


Fig.16.5: Micrograph of a laser-welded ferrule

Pull tests have shown that it is generally not the welded joint itself, but the POF coat which breaks. Figure 16.6 shows the schematic view of the structure of such a welding fixture (welding unit with manually operated fiber feed).

In this machine, two fibers are processed simultaneously in order to improve the throughput. Both fibers are welded to the ferrule in one step. The ferrules are fed to the actual welding point from two rolls as an endless tape. The operator must insert the fibers into the feeds of the fiber ends and start the welding process manually.

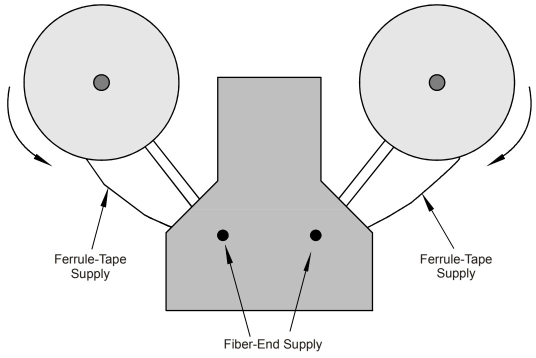


Fig. 16.6: Structure of a semi-automatic laser welding fixture

The plastic of which the ferrule is made must be suitable for laser welding. Currently only one material is known and available on the market. It is a polyamide 12, which is sold in different versions under the product name *Grilamid*. It contains a certain percentage of glass and can also be colored without impairing laser weldability.

The manufacturing process must ensure that the fibers keep a defined backward clearance of greater than or equal to 0 mm with regard to the end face of the ferrule. This is necessary in order to reliably prevent a collision between fiber and counterpart. For MOST150, the clearance at the side of the header is defined by the MOST150 FO-Transceiver (SMD-header) drawing to be 0.01 to 0.1mm behind the ferrule endface. For MOST25, the clearance is within the scope of the respective manufacturer. At the side of the connector, the MOST Specification defines the values -75 µm +/- 25 µm. A pin which is situated in the ferrule holding fixture of the laser welding unit and advances by the corresponding value ensures that the required clearances are kept with sufficient accuracy.

In general, one tries to keep the attenuation of a fiber as low as possible. There are, however, special cases in which a pre-defined optical attenuation must be introduced. This is ensured by way of an additional backward clearance of the fiber with regard to the end face of the ferrule. Compliance with the value of the backward clearance is ensured by means of a pin advancing by a respective value in the laser welding unit, analogous to the fibers with a standard attenuation.

The ferrules are delivered in rolls as an endless tape. Both welding and cutting are conducted on the laser welding unit, i.e., the ferrule is cut from the endless tape and the burr is reduced to an admissible value.

Testing

If the sets of fibers are manufactured and processed separately from the header, it must be ensured by special testing measures that their attenuation is not higher than a defined limiting value. MOST150 gives an attenuation limit of 1.5 dB for the pigtail fiber. Thus it is ensured that even a combination of a header with the worst permissible performance and a set of fibers with the highest permissible attenuation still comply with the permissible tolerances for output radiation efficiency and responsively according to the MOST Specification. Otherwise it would be necessary to conduct an end-of-tape selection test for the overall layout of the set of fibers and the header with regard to radiation efficiency.

In addition, random testing using a pull strength measurement setup must ensure that the pull-off strength at the welding point is sufficient.

The values for the pull-off strength are found to be very constant. Additionally, the mechanical stress is very low, in contrast to a vehicle wire harness.

With regard to the optical attenuation of the fiber pieces, the following influencing factors are known or possible:

- Fresnel losses (~0.2 dB/end face, ca. 0.4 dB per connector)
- Axial displacement of the fibers relative to each other
- Radial displacement of the fibers relative to each other
- Slanting position of the fibers relative to each other

- Damaging of the cladding due to errors in the laser parameters and variations in the thickness of the fiber coating
- Optical quality of the saw face / condition of the saw blades

Fresnel losses occur at each transition from one medium to another. They add up to ca. 0.4 dB per connector.

The maximum axial displacement is defined by the MOST Specification (connector side) and by the manufacturer's specification (header side).

A measurement setup in the laser welding device ensures that the value for each manufactured fiber be within the permissible mechanical tolerances.

The radial displacement results from the diameter tolerances of ferrule and fiber and from the maximum eccentricity of the POF coating.

The slanting position of the fibers relative to each other may result from a tilting within the connector or from a slanting position of the fibers during the sawing process. The latter is prevented by a correct setup of the sawing fixture. The first is only very small for constructive reasons.

The wrong power setting of the laser in the welding unit in combination with unavoidable variations in the thickness of the coating can theoretically cause damage to the POF cladding. This could result in higher attenuation.

Experience has taught that the condition of the saw blades is the one factor influencing the attenuation of the fibers that has to be supervised most closely during manufacture. The other parameters are subject to only slight variations or can be virtually neglected. As mentioned before, this is achieved by exactly defined maintenance intervals and the 100% testing of the fiber attenuation.

For determining the fiber attenuation, the *insertion loss method* according to IEC 61 300-3-4 is suggested by the MOST150 Measurement Guideline. The test setup is calibrated periodically by measurement without DUT. The DUT's attenuation is the difference of the measured optical power with the DUT included and not included.

The *substitution method* according to IEC 61 300-3-4 is also used as testing method for determining the fiber attenuation. The test set up is calibrated periodically by means of a reference standard (Golden Fiber). This is a fiber with exactly defined attenuation capacities and structure. The underlying measurement setup can be gathered from figure 16.7:

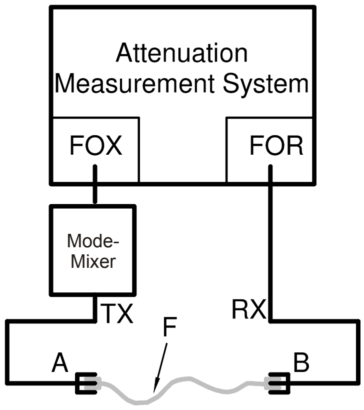


Fig. 16.7: Basic setup for measuring the attenuation of a fiber

Fiber F to be tested is clamped into test contacts A, B in a calibrated measuring device and the measurement process is started. By using an attenuation measuring device, it is compared to a fiber with known characteristics. Thus the respective attenuation is ascertained.

The following apply for both measurement methods: The radiation is induced into F according to IEC 793-1-1. The output signal of the FOX is conducted through a mode scrambler. This ensures that there is the same angular distribution and intensity density allocation at its output as can be expected at the end of a “long” POF stretch. This is necessary to ensure comparability of the measurement data gained and independence of the radiation source used (dimension and geometry of the bright spot, angular dependence).

16.1.3 Connectors

A device is connected externally via the device connector/s to the wire harness. The nomenclature (2+N and 4+N) used for the connectors indicates that they are suitable both for connecting electrical as well as optical signals to the respective device. In order to ensure compatibility, the MOST Standard defines the interface geometry towards the wire harness. The interior design of the device may vary, however. The 2+0 connector for MOST150 is an exception to that. Here the footprint and maximum dimension of the connector is defined within the physical layer specification.

Figure 16.8 shows a 2+4 connector for use with flexible pigtails. At the right edge one can see the four electrical contacts. The two ferrule holes accommodate the MOST ferrules.

2+0 connectors are usually only used with rigid pigtails, although a 2+0 connector with a flexible pigtail would be possible. In contrast to this, connectors with optical and electrical contacts are mostly implemented as flexible pigtails (see fig. 16.8).

Consequently, all connectors at the device have electrical contacts – either the electrical connections of FOX and FOR, or any signal lines required within the MOST component. Thus all connectors have to be run through the soldering process.

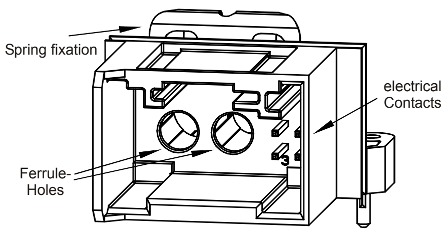


Fig. 16.8: 2+4 device connector

16.2 Processing within the Scope of Electronics Manufacture

Opto-electronic components are more delicate with regard to processing during soldering than other comparable electronic components. This is particularly the case with MOST components if not only FOX and FOR but also the POF must pass through the thermal process during soldering. The SMD-header for MOST150 was specified to break up the pigtail into separate parts: one part is the header designed for automated assembly, the other part is the pigtail fiber set that can be kept out of thermal processes.

16.2.1 Mounting and Soldering of Headers

THM based headers

Headers based on FOR/FOX in THM packages are components that typically cannot be mounted by means of a mounting machine. They must be positioned manually before the soldering process. The header may be mounted by machine if it is guaranteed that the position of all THM-pins of the header is within the overall tolerance range of the PCB’s pinhole position and diameters without overlap. If the pigtail fibers are already premounted and are thus led through the soldering process, precautions have to be taken to avoid damage. With regard to the protection of the fibers and the handling convenience, it is better if the internal set of fibers is mounted only after the thermal process in the soldering bath.

For wave soldering and partial soldering the assembly is usually first preheated using infrared radiators on the soldering side. This is required in order to avoid thermo-mechanical stresses. Temperatures of up to 100 °C are reached at the soldering side. The assembly then passes over the soldering wave.

The housing and the pins of the FOX and FOR, as well as the sheet metal case of the exterior shielding are heated up significantly during this process, even though immersion into the soldering bath only takes a few seconds. A soldering bath has a temperature of 240 to 260 °C. If a lead-free soldering process is used, these values are about 20 °C higher, because the lead-free solder has a higher melting point. The FOXs and FORs used must be specified to be processed at such high temperatures. Due to the structure and the optically transparent plastic material used, damages to the assemblies may be caused by high thermo-mechanical stresses. The plastic material and the leadframe have different coefficients of thermal expansion. The heating of the material nearly reaches the glass transition temperature of the potting compound used. In this range, the coefficient of thermal expansion changes considerably, which results in additional stresses. Unfavorable soldering profiles may cause delaminations and debonding.

In the case of premounted sets of fibers, it must be ensured by measuring that the fiber end faces do not heat above the maximum temperature permissible for the optical wave guide. Characteristic damage patterns are concave vaultings at the fiber end face. In such a case, the fiber attenuation rises significantly.

Usually, the transparent potting compounds used here are also hygroscopic. A typical humidity class of the FOTs is 2A. Long-lasting, unprotected storage should thus also be avoided. If necessary, the components will have to be dried before processing.

SMD headers

The header package for surface mount technology, as defined in the MOST150 physical layer specification, is typically delivered within tape-on-reel. It can be processed by standard pick-and-place equipment as commonly used in electronic assembly lines. The SMD-header is placed in a pocket of the packaging-tape which is furled on a reel according to DIN EN 60286-3:2008-4. Typically, the SMD-header is delivered with a Kapton™ tape on top providing a flat and clean surface for vacuum pick up nozzles. Thus it can be picked up and placed on the printed circuit board just like any other SMD component of similar size and weight.

16.2.2 Mounting and Soldering of Connectors

In line with the headers, device connectors are components that have to be positioned before the soldering process. That applies to both the standard connectors with an optical interface and to µPigtails with and without additional electrical contacts. Due to their geometry, these components cannot be effectively mounted automatically.

The soldering of the standard connectors is not a critical process. They are passive elements with a high thermal stability. Rigid pigtails have to be treated differently, as was mentioned before with regard to the headers. In this case the FOXs and FORs used must be specified to be processed at the respective soldering temperatures. Both the plastic lenses and the short fiber pieces used for bridging the gap between the FOX or FOR and the set of fibers are exposed to high processing temperatures.

16.2.3 Mounting of Internal Sets of Fibers

There are pigtail types in which the internal sets of fibers are processed separately from the headers as it is standard by concept for the SMD header, and others in which the set of fibers and the header already form a unit when they are delivered. The latter require specific handling for processing. The connection of the set of fibers at the end of the tape is, however, easier. If the set of fibers and the header are mounted separately, the fibers are mounted at the header side and at the connector side after the last thermal process. The holding spring for the ferrules at the connector side not only ensures fixture, but also the springing behavior required by MOST. Previous to mounting the fiber on the SMD header side, the Kapton™ tape on top of the SMD transceiver needs to be removed. The fibers can then be plugged into the respective cavity. To prevent mixing up receive with transmit path, the FOT-cavities and fiber ferrules are coded by diameter. The cylindrical part and the flaring ring on the ferrule, both contribute to the coding. Their diameters are inversely different on receive and transmit ferrule. The flaring ring on the ferrules provides the bearing surface for the clamp mechanism integrated in the SMD header. Fixation of the pigtail fiber on the SMD header is done by a snap in mechanic that holds the ferrule with a certain force within the header cavity after the plug in force has been exceeded. To remove the fiber, a higher pull-out force is needed.

16.2.4 Device Testing

At the end of the manufacturing steps, an additional final test of the MOST device is conducted. In contrast to the previously described tests, this test is conducted on the physical and the logical level. The test aims to verify the reliable function of the device and to detect possible errors from the manufacturing process.

By suitably choosing the stimulating signals (radiation power, signal curve form), the system reliability at low input signal levels can be tested. Additionally, the power of the FOX is tested at the optical outputs for compliance with the requirements of the MOST Specification. Characteristic error patterns are complete failures of FOX or FOR, insufficient transmission level at the TX output, or an early rise of transmission errors at low signal levels.

This procedural method ensures that devices that are delivered to a customer have no problems with MOST communication.

16.3 Summary

This chapter deals with the manufacturing and processing of opto-electronic MOST components. It explains the differences with regard to the processing of conventional electronics components and points to the demands made on the assembling and testing methods.

Several years of experience with MOST25 in the field have proven that the methods described no longer pose any practical problems. They are well-established and stable.

With MOST150, the demand for an increased data rate has been answered but may be not satisfied for all times. The SMD FOT gives an answer on the demand for components that simplify handling with regard to manufacturing. It can be placed by a machine and soldered within the reflow soldering process potentially eliminating the need for selective wave soldering. Due to its nature of being a true surface mount component, the area on the opposite side of the PCB is not blocked for other electronic components. Thus valuable PCB area is deallocated by using an SMD header.

17 Gateways to MOST

Several communication systems are often used and connected via a central gateway in a car. Signals interact via this gateway.

17.1 Distribution of Signals via a Gateway

Figure 17.1 shows an example of the distribution of signals in a vehicle network. The ABS sensors generate speed sensor pulses depending on the speed of the wheels. The ECU converts the pulses into the signal *WheelSpeed,* which is made available to all ECUs connected to this CAN and to the gateway via the Powertrain CAN. This kind of transmission of information is based on the Publisher Subscriber Model. The sender (publisher) cyclically sends its data and the receivers (subscribers) read the data from the bus if required. In the example given, the central gateway receives the data and transmits them to the Infotainment CAN. There, the data are transferred into the MOST system by the HMI, which implements the gateway functionality. The signal *WheelSpeed* is used, for example, by the navigation system for correcting the navigation data, or for detecting the vehicle position in tunnels where GPS signals cannot be received (dead reckoning). The radio, too, needs the information for adjusting the volume. In the MOST system the signal *WheelSpeed* is converted to a property of the FBlock *Vehicle*.

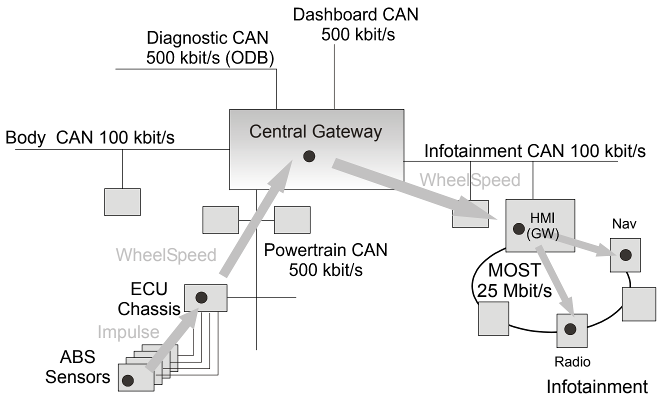


Fig. 17.1: Example of a signal transmission from the ABS sensor into the MOST system

Whereas the Powertrain CAN uses a bit rate of 500 kbit/s, the Infotainment CAN uses a bit rate of 100 kbit/s. A CAN – MOST Gateway is integrated into the Head Unit (Human Machine Interface, HMI). As shown in figure 17.1, the signal *Wheel-Speed* must be converted both, in the central Gateway and in the HMI. A bit rate adaptation must be performed in both gateways and, additionally, a protocol conversion in the HMI.

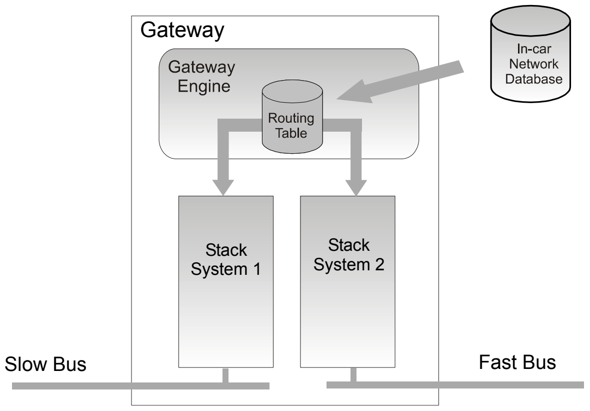


Fig. 17.2: Basic structure of a Gateway

17.2 Architecture of a Gateway

Figure 17.2 shows the basic architecture of a Gateway. Usually, the complete stack, comprising all seven layers of the OSI communication model (Open Systems Interconnection Reference Model), must be implemented for each bus system. The Gateway Engine is placed above, routing the signals from one system to the other on the basis of the static Routing Table. The Gateway Engine lists all signals with source and target and is generated from the central In-car Network Database of the OEM. The following are the tasks of a Gateway:

- Address conversion
- Message conversion
- Intermediate buffering of the messages
- Packet confirmation
- Flow control and bit rate adaptations
- Routing of the signals
- Network management
- Error management
- Locking/separating of the buses, if required (e.g., flashing/programming of two buses at the same time)
- Authentification of message senders/receivers (e.g., only ECU x in MOST may send to ECU y in the CAN)

The listing shows that the implementation of gateways is usually complex and error-prone. The different bit rates of the networks can result in delays and loss of messages.

The structures mentioned will now be explained in detail, taking the MOST – CAN Gateway and the MOST – Bluetooth Gateway as examples.

17.3 MOST – CAN Gateway

The MOST – CAN Gateway is implemented in the HMI or in the central gateway. Figure 17.3 shows the structure. There are different levels of abstraction for the CAN bus and the MOST Interface. Whereas there are signals that are transported to a specified position in the CAN frame, the MOST Interface consists of a function block, i.e., the signals on the MOST side are treated as properties of the function block. All important signals that have to be exchanged between the MOST system and the rest of the vehicle network are represented in one FBlock, e.g., *Vehicle* [FBVehicle 1.6] or, when reflecting also the source domains of the information, in multiple FBlocks, e.g., *Powertrain*, *BodyCompartment, HumanInterfaceController* and so on.

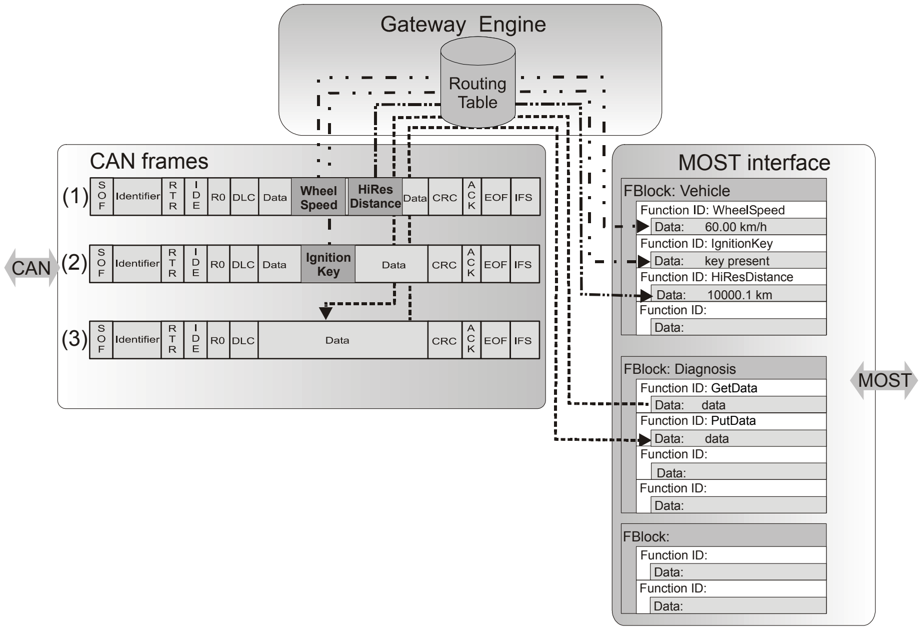


Fig. 17.3: Structure of the MOST CAN Gateway [according to Por 2004]

In the example given in figure 17.3, the signals *WheelSpeed* and *HiResDistance* (high-resolution odometer reading) are copied from the CAN frame 1, and the signal *IgnitionKey* (status of the ignition lock) from the CAN frame 2 onto the corresponding properties of the FBlock *Vehicle*. The MOST devices interested in these properties can subscribe to these properties by using the notification mechanism at the FBlock *Vehicle* in the gateway.

If a diagnostic protocol is used in the gateway, flow control must be implemented due to the different bit rates.

The opposite direction, where signals are originated on MOST and need to be gated on CAN, the Gateway is instantiating one or more Shadows. These Shadows are keeping themselves registered for notification, in order to get posted upon status changes on the properties of the related FBlocks.

A Protocol-Gateway therefore is—from the perspective of any other native MOST element, FBlock or Shadow in the ring—following all MOST stereotypes in a fully MOST Specification compliant way.

17.4 MOST – Bluetooth Gateway

The Hands Free Profile [HFP 1.5] was defined for connecting a cell phone with a headset or the telephone unit in a car. The profile comprises the role of the Audio Gateway (AG), which is usually adopted by the cell phone, and the role of the Hands Free Unit (HF), which occupies the headset or the MOST Bluetooth Gateway in the car (fig. 17.4).

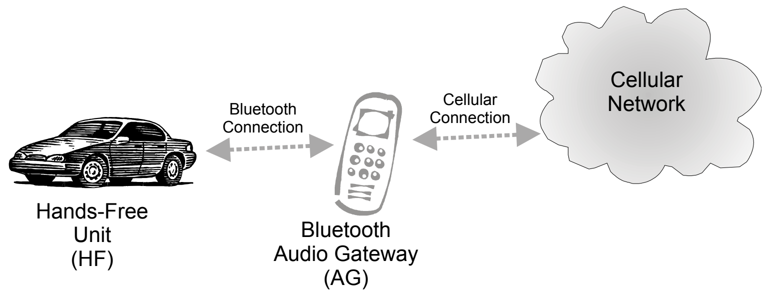


Fig. 17.4: Roles of the Hands Free Profile [according to HFP 1.5]

In addition to the Control Channel, the audio channel also has to be routed in the Bluetooth Gateway.

17.5 Routing of the Control Channel

Figure 17.5 shows the stack architecture of the Control Channel in a MOST – Bluetooth Gateway. It becomes clear that Bluetooth and MOST have a similar architecture. While the FBlock *Bluetooth* is placed on the MOST Stack, Bluetooth uses the profile *Hands Free Control* for controlling the Hands Free Unit. It basically standardizes the *Call Handling*. This comprises taking calls and setting up and terminating the audio channel (see also appendix B 2.5). The Gateway Engine connects the services of the Bluetooth profile with the properties and functions of the MOST function block *Bluetooth*.

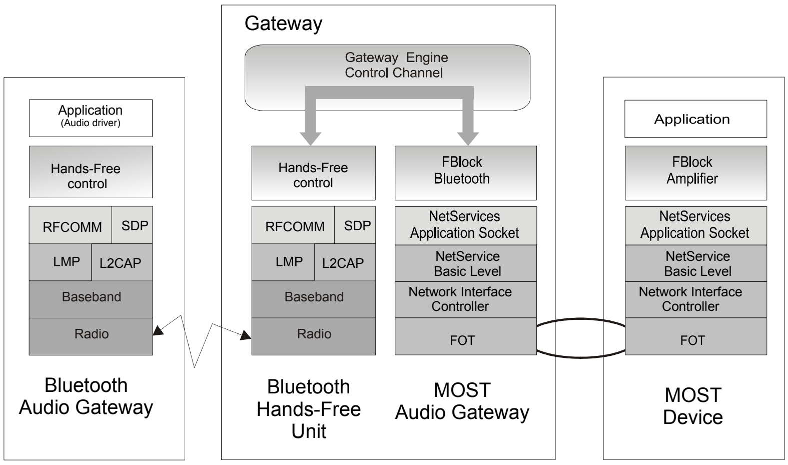


Fig. 17.5: Architecture of the Control Channel

Figure 17.5 shows the simplified structure of a possible audio connection setup in the case of an *incoming call*. The audio connection setup is initiated by the HFP (AG). The information about the successful setup is then available in the HFP (HF) and is passed on to the Gateway Engine. It is only now that the audio setup via MOST takes place. For that purpose, the Gateway Engine calls the method *BuildSyncConnection* of the FBlock ConnectionManager (CM), which then sets up the synchronous MOST connection between the Bluetooth Gateway and the amplifier by using the methods *SourceConnect* and *Connect*. Using *BuildSyncConnection.ResultAck*, the ConnectionMaster reports the successful setup to the Bluetooth Gateway. At the same time the ring tone is created in the MOST system.

Entries in the phone book can be exchanged between the cell phone and the MOST system by means of AT commands or the Synchronization Markup Language (SyncML), if the latter is implemented. Another possibility is the Phone Book Access Profile (PBAP) defined for that purpose, which then has to be implemented in the cell phone and in the gateway [PBAP].

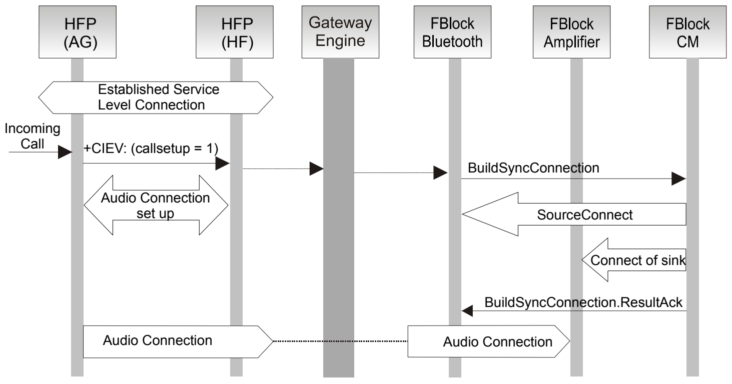


Fig. 17.6: Basic sequence for the setup of a synchronous connection

In cars, the SIM Access Profile (SAP) is also of interest [SAP]. It is used exclusively for exchanging the SIM data between the cell phone and the car. By means of the SIM Access Profile, the telephone integrated in the car can read out the SIM data from the cell phone and use them as long as it is registered via Bluetooth. This makes a second SIM card unnecessary.

17.5.1 Routing of the Synchronous Audio connections

In addition to the Control Channel, the synchronous audio connection also has to be routed. In this context, the frequency spectrum and, if necessary, the amplitudes have to be adapted. The Bluetooth SCO channel is adapted to the audio quality of an ISDN channel, i.e., it has a sample rate of 8 kHz, which allows a signal spectrum of up to 4 kHz.

Bluetooth uses the Pulse Code Modulation (PCM) or the Continuous Variable Slope Delta Modulation (CVSD) for source coding of the audio signals on the SCO channel.

The PCM format can be used with a logarithmic A-characteristic (A-law) or a logarithmic μ-characteristic (μ-law). As the sampling values are quantized with 8 bits and the sample rate is 8 kHz, a bandwidth of 64 kbit/s is required on the SCO channel.

The CVSD modulation transmits the differences between the anticipated and the actual sampling value. It is a type of delta modulation, in which the sampling period of the approximated signal is incremented or decremented progressively, in order to improve adaptation of the signal to the analog input signal. For the conversion, only the positive or negative changes compared to the preliminary value are indicated by a bit. The sample rate is 64 kHz and the amplitude is 16 bits. As only one bit is transmitted at a time, the channel also requires 64 kbit/s.

Usually PCM modulation is used for Bluetooth. The adaptation of the different sample rates is effected by using a bi-directional asynchronous sample rate converter, as is shown in figure 17.7. In the case of audio data transmitted from the Bluetooth channel into the MOST system, the converter increments the sample rate to 44.1 kHz by means of interpolation. In the opposite direction, it decrements the sample rate to 8 kHz. A low pass filter is integrated into the converter.

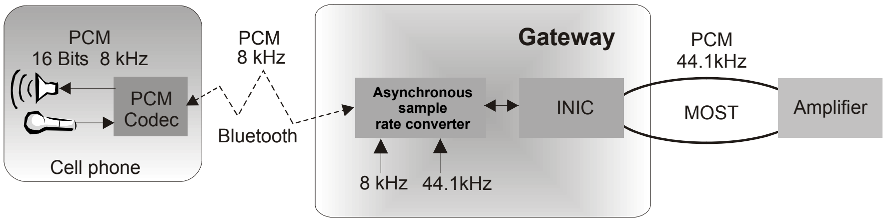


Fig. 17.7: Architecture of the Audio Gateway

The controls of the Streaming Channel and the Control Channel must be coordinated by the gateway. A valid signal must first be applied to the synchronous connection, for example, before the amplifier switches to the telephone source. Otherwise whistling or other noises might occur.

18 MOST Applications

This chapter deals with an overview of some MOST based applications, starting with a glance on the history of infotainment systems and an outlook into the future from today's perspective.

18.1 Overview

Back in the mid 90's, the automotive entertainment market was confronted with new **technical and operational requirements**:

1. The growing number of entertainment components supplying and consuming all kinds of information in the car increased the complexity of interconnecting and controlling such components.
2. The car makers struggled with Electro Magnetic Interference (EMI), as well as the weight and cost of the wiring harnesses, caused by the multiple different hardware platforms. The diagnosis of the different components could not be inherently handled by the system, but only through each individual component.
3. The final user (driver or passengers) had and still has the need to operate all the components through different, individual human machine interfaces.

To build a system, each component (whether a sink or a source of information) has to have the capability and possibility to communicate with other components in such a way that the information can be used in real time (minimum latency). Furthermore, the information sometimes is enhanced and post-processed (synchronous co-processing), and re-used again as a new set of information. The infotainment Head Unit, sometimes also named 'system controller' should govern all events in all components in such a way that the user is no longer confronted by the complexity of the system, but has to deal with the results produced by the system only!

This set of requirements lead to the design of the MOST system, enabling the transport and 'broadcasting' the information (data) and the control commands in such a way, that media-signals could also be transported in real time.

MOST quickly became the backbone of all modern infotainment systems and is now the de facto standard for infotainment networking in the automotive industry. However, MOST does not only stand for the transport of the data. Equally important is the fact that MOST offers a standardized set of APIs enabling the design of an appropriate software structure around the logical representation of all components (proxies) in the system controller.

And last but not least, MOST was and still is the enabler to overcome the real challenge in infotainment system design: Provide the flexibility and the scalability to cover different market requirements with different levels of functionality and performance based on a unique but still stable system architecture.

The FIRST Generation of MOST based Infotainment Systems in the market can be characterized by a strongly distributed architecture, driven by a large variety of infotainment functions and features with limited and at the same time expensive computing horsepower. The result was a pretty strongly modularized approach with many dedicated system components carrying dedicated functions. The system components were designed and manufactured by several suppliers, represented by their individual, sometimes diverse skill sets in hardware and software design and manufacturing. Hence, the difference in quality and performance of the components was high, until a later stage when everybody had caught up in the learning curve. Figure 18.1 shows such a system.

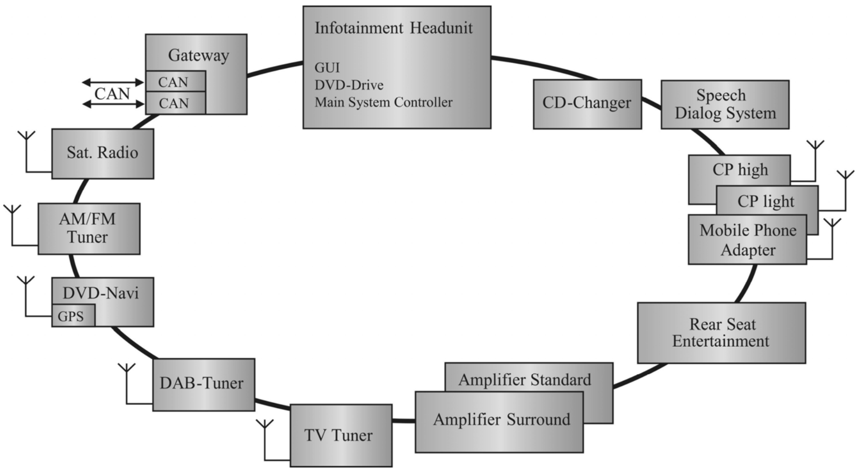


Fig. 18.1: 1st Generation MOST based Infotainment Systems

The SECOND Generation of Infotainment Systems is characterized by a higher physical integration of the external components into the headunit. The number of components could be typically reduced from 12-15 to 6-8. Cost, weight and power consumption were typically down by up to 30%, compared with 1st generation systems of similar functionality and performance. Examples for such physical integration are 'Tuner-Amplifier' modules or the integration of navigation computing modules into the headunit (see fig. 18.2). The field results of the 1st generation MOST cars (BMW's 7 Series E65 or Daimlers E-Class W211) vs. the 2nd generation cars (BMWs 5 Series E60, MB C-Class W203) are the proof of this.

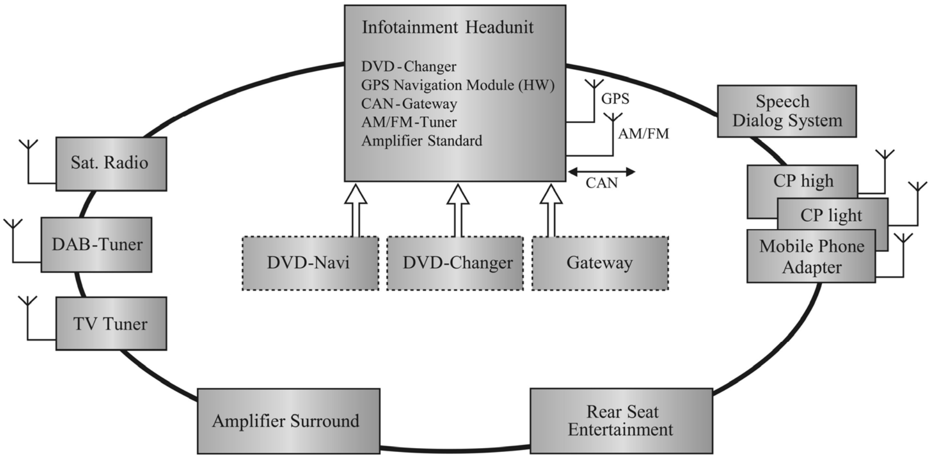


Fig. 18.2: 2nd Generation MOST based Infotainment Systems

The THIRD Generation of Infotainment Systems are the basis for the systems established on the market from 2005 onwards. They are characterized by the replacement of hardware components through software modules (see fig. 18.3). Further cost, weight and power consumption reduction are achieved (-30%). Many system functions are absorbed by the Head Unit and a lower number of external system components. In some cases, the entry level infotainment system does not even contain any external system components anymore, while through the built-in MOST interface still offering the possibility to attach advanced options such as external sound amplifiers, media changers or country specific tuner options like satellite radios.. Here, devices identical to those in the mid and high-end system can be used.

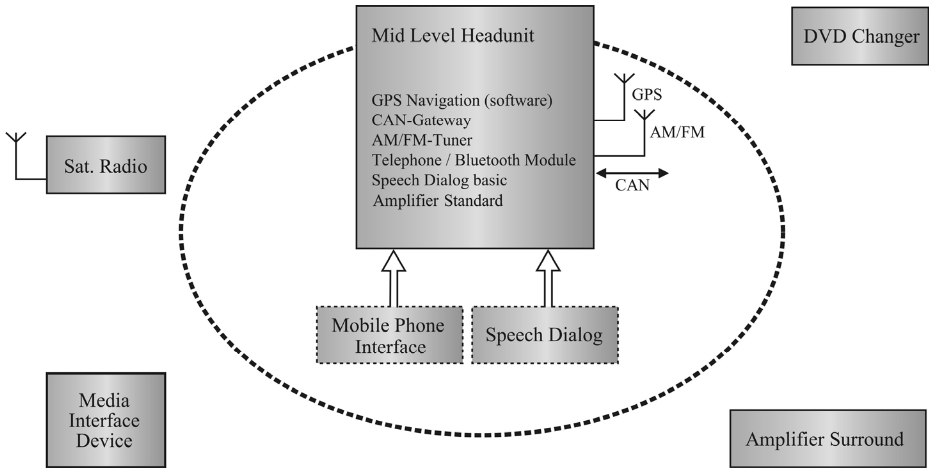


Fig. 18.3: 3rd Gen Mid Level Infotainment System

The FOURTH Generation of Infotainment Systems will be seen in the Market from 2013 onwards. With the functions and features of MOST150 and particularly its enhanced bandwidth, it will finally be possible to transport all infotainment related video data over the MOST network. The video content is transported in a compressed way, as all media data is provided in a compressed format anyway (over the air through digital broadcast or wireless LAN, on DVD or Blu-ray discs or in the memory of portable media).

Typical systems will comprise an infotainment headunit and some external components like sound amplifier, digital tuner (country/region specific), media interface, and rear seat entertainment units or the inststrument panel cluster (IPC), which in times of LCD panels, are gaining more and more video capabilities. Optionally, a back-up camera, which as of today is still connected to the headunit via separate video cable, may be integrated into the MOST system (see fig. 18.4).

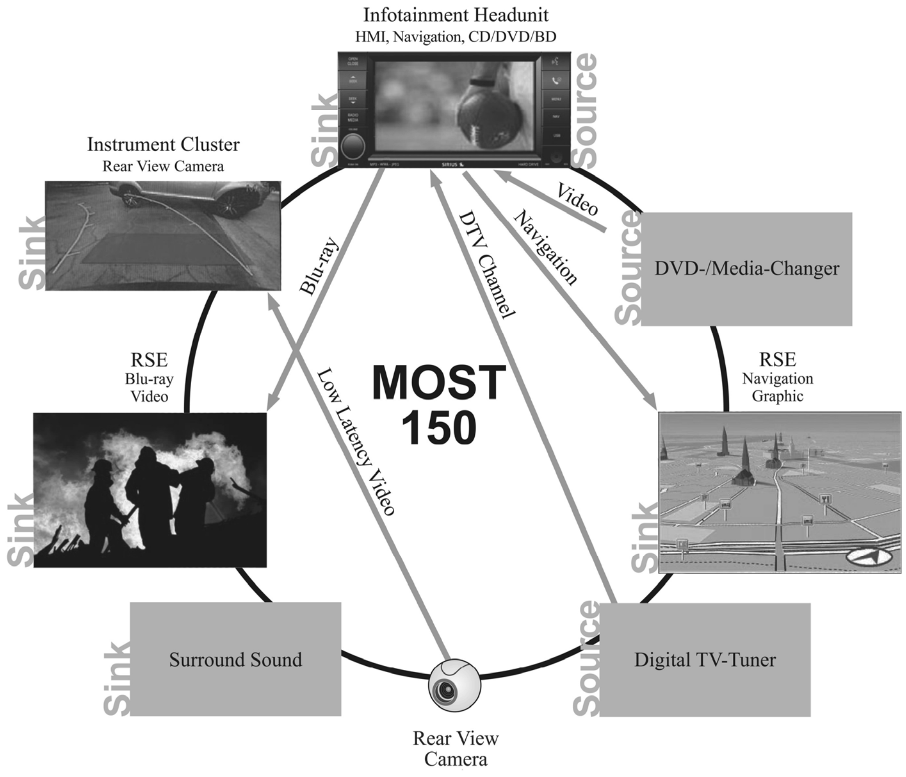


Fig. 18.4: 4th Generation MOST based Infotainment System

The following three sections will elaborate on three examples of MOST system components: infotainment headunit, media interface and sound systems.

18.2 Infotainment Headunit

In 2010's systems, a typical infotainment headunit is a scaleable computing platform optimized for the harsh automotive environment with computing power reaching from 32-bit based 400 MIPS up to 2 GIPS dual processor architectures. In addition, dedicated hardware (DSPs) for Audio/Tuner and graphic processing is used. State-of-the-art 32-bit operating systems are applied like Microsoft WinCE4Automotive, QNX, embedded Linux or Micro-Itron.

Accordingly, a respective software suite is required to be matched to the scalability of the hardware. Innovations concerning minimum start up time, real-time processing, processing streaming data, server capability of the modules, multiple graphic drivers for multi-user applications and multimodal HMI operation are realized through advanced application software modules, advanced frameworks and real-time operating system (RTOS). It is extremely useful to implement pervasive software architecture from entry level up to high-end systems, following the MOST philosophy.

In order to guarantee fast startup of the individual system components, which is very important for providing the functionality of all system components to the user quickly, very often an additional 16-bit microcontroller, often called I/O processor, with a different, lean OS is used. This device also typically implements the gateway functionality (MOST-CAN), in case the headunit carries the gateway.

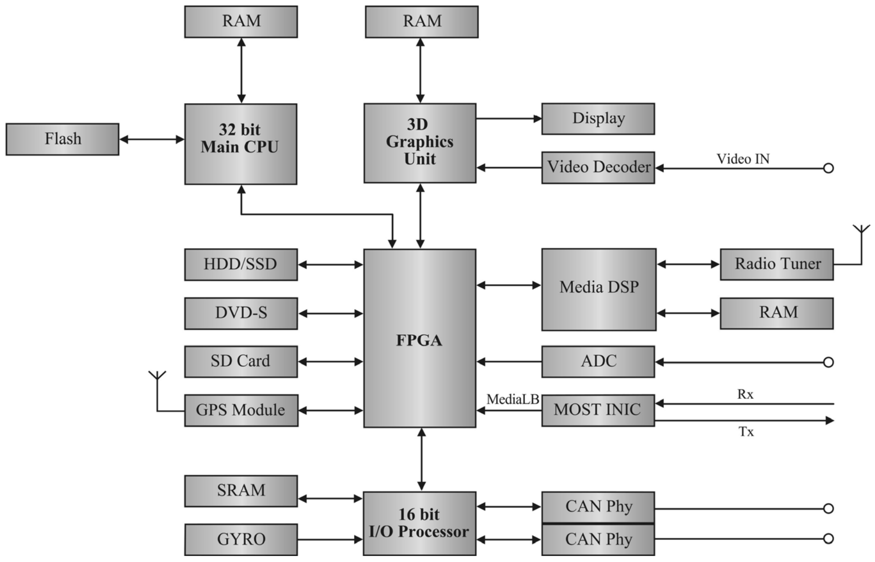


Fig. 18.5: Infotainment Headunit based on FPGA architecture

As MOST defines a comprehensive set of data types to exchange information 'signal based' and includes an expandable and customizable set of already defined APIs, a very interesting approach consists of a unified communication architecture for a): the communication over the physical MOST network, b): the communication between the main CPU and the I/O processor and c) the 'on chip' communication inside the 32-bit engine. The majority of the MOST based headunits from the 1st and 2nd generation MOST systems have been using such SW architecture, however at the time being pretty much proprietary solutions.

Today, SW frameworks offering ready-made solutions for such software architectures are widely used and commercially available, e.g., from K2L (www.K2L.de)

Market leaders in field-programmable gate arrays (FPGA) have committed their next generation devices to support scaling the main board and interfacing hardware in a very flexible and optimized way. The advantage of such a concept is a scalable approach providing a number of hardware levels, which can be re-configured by the FPGA code through the system software (see fig. 18.5).

A different approach is shown in figure 18.6. It is the even stronger re-use of PC-like architectures. In this case, the FPGA is replaced by a dedicated south bridge chip (I/O Hub), similar to what can be found in PCs. However the south bridge chips used in PCs cannot be used due to missing Audio/Video ports. Here, the still significant difference between car applications (embedded multimedia units) and PC applications becomes obvious, forcing the development of automotive specific I/O hubs.

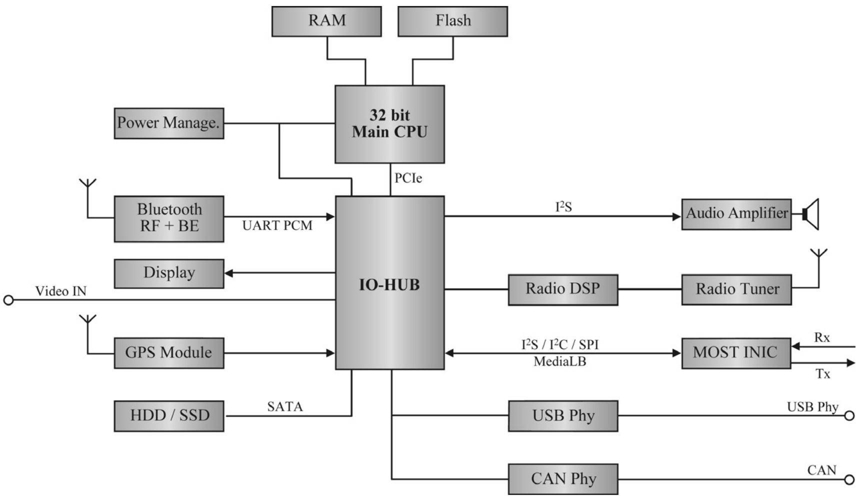


Fig. 18.6: Infotainment Headunit based on PC architecture

It has to be seen if and until when the complete integration of the multimedia (A/V Processing) functionality into the main processor will be realized. In any case, such an integration will require the existence of a comprehensive set of software suites starting from RTOS and framework, offering different packages of well debugged software suites, incl. applications as network subsystems (MOST), graphic libraries and drivers, navigation, voice control and voice enhancement, multi standard media decoders, discrete and enhanced sound processing, digital radio processing, generic system diagnosis, system start-up and shutdown including the network control, etc.

As of 2010, different initiatives exist to create such an ecosystem. However, the migration is quite challenging from a performance, scalability and design/software integration perspective, particularly given the constantly and rapidly growing feature set of mobile/portable devices such as smartphones and media players, which are driven by the much higher volume consumer industry.

Another trend which may become even more relevant for the design of future Infotainment headunits is the emerging requirement for driver assist functions and features. As the infotainment system provides the main user interface, it is obvious that there will be some overlap between the classical infotainment domain and the upcoming driver assist cluster, which will also affect the infotainment headunit design and architecture.

Given the fact that both non safety related as well as safety related functions need to be considered, a more modular approach may be even better, allowing the development of different parts of the headunit according to different Automotive Safety Integrity Levels (ASIL) according to the ISO 26262 standard.

In any case it is obvious that a powerful communication backbone providing different levels of communication methods, as MOST provides it, becomes even more important than before.

18.3 Media Interface Devices

The variety in media devices is constantly expanding and the design of the devices is rapidly changing. MP3 players (the most prominent one being the iPod) using different APIs and sometimes also copy protection/content management methods require more or less frequent changes on the vehicle side. Some companies even require dedicated hardware (ICs) to authenticate and enable the communication with their respective products.

Implementing such changes in the multimedia headunit implies a relatively large development and in particular test and requalification efforts. Other media devices may not even be available during the time the headunit and the multimedia system is developed, or may even emerge only during the production time of the car. From a car makers perspective, this discrepancy of the vehicle lifecycle and the consumer products has become an ever increasing nightmare.

For obvious reasons, it does make a lot of sense to de-couple the rapidly changing consumer world from the more stable car environment. This is often achieved through so-called ‘media interface devices’ or sometimes called ‘universal communication interfaces’. Such devices may contain different types of card readers, multiple USB interfaces, and, respective authentication devices, to communicate with specific, proprietary equipment. Figure 18.7 shows the block diagram of a media interface device.

As discussed in the previous section, today’s smartphones carry more and more functions and features: navigation, voice control and, multi standard media servers and wireless communication including GSM/UMTS and also WLAN. The demand to attach such devices to the car and use the built-in features is constantly rising.
Using a MOST based media interface device, the car maker has also the flexibility to react on fast market changes coming from the consumer industry on the one side while still being able to keep the well defined, carefully architected and debugged infotainment infrastructure.

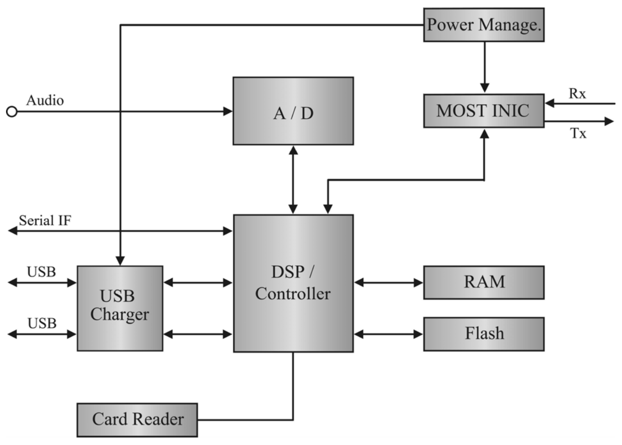


Fig. 18.7: MOST based Media Interface Device

18.4 Sound Systems

Quite often manufacturers offer several optional equipments for each vehicle type. Different trim levels, reaching from basic systems up to (often branded) premium sound systems, are available. It is obvious that, although offering different features and in particular output power and signal processing capabilities, it is extremely important that all these trim levels can be controlled in a consistent way from the network side. A unified control API as provided by MOST is the key to a clean and straightforward system architecture and integration process.

Speed-dependent volume control is one example, where the sound system needs to be networked to the rest of the vehicle. It compensates for disturbing vehicle noises that swell with increasing driving speed.

Another example is the generation of acoustic warning signals (chimes), e.g., for park distance control or seat belt warnings. With growing feature sets of driver assist applications, where the driver is more and more overloaded with visual information, it can be expected that the acoustic information channel to the driver will get more important. Such signals are transmitted to the Head Unit via the MOST or CAN bus system, where the real-time behavior is extremely important.

Premium System

In a *premium system* a multitude of signals is transmitted from the Head Unit or other sources via MOST to a separate DSP (Digital Signal Processor) amplifier, which contains multiple amplifier outputs. This amplifier is also referred to as digital sound processor (see fig. 18.8).

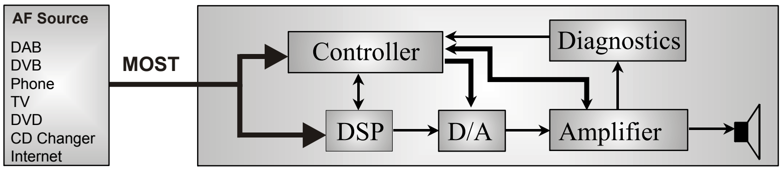


Fig. 18.8: Block diagram of a DSP Sound Amplifier

The DSP used does not only allow for a broadband, speed-dependent volume level increase, but also influences the spectral reproduction characteristics by counteracting the masking influence of low-range noises by a low-frequency level rise at the wanted signal. Noise in a vehicle is mostly very low-frequent and decreases by approximate 20 dB per frequency decade. Thus, a simple broadband level rise of the wanted signal – as in the case of the premium system – leads to a high-range impression, because only low frequencies of the wanted signal are masked by the noise and are thus less audible. Due to the fact that noise depends on the speed, it can be estimated via the rev counter or be measured directly by a microphone. This results in a high-grade closed-loop control system. Typically, the microphone signal (one or multiple microphones may be applied) is also transmitted via the MOST bus.

18.5 Audio and Video Transmission Formats

18.5.1 Audio Streaming

A multimedia system in a vehicle must be able to read the different storage media and their coding formats. In addition, the different transmission formats of digital audio broadcasting and digital video broadcasting must be identified. Depending on the equipment variant, high transmission rates may be generated, which must be processed by the MOST system.

In contrast to the vast majority of other bus systems coming from different application areas, MOST has been designed for Multimedia Data Transport throughout a system, as its name already indicates.

In contrast to home systems, the car requires a multi source / multi sink environment, where multiple audio and video sinks, amplifiers, headphones and multiple displays are operated at the same time.

There is a great variety of transmission formats in the audio field. In addition to stereo reproduction, different surround formats are relevant for the use in vehicles and are often employed in professional systems. The following is a rough overview:

Stereo

The uncompressed signal has a data rate of 1.4 Mbit/s at a sampling rate of 44.1 kHz and a 16-bit resolution. The most widely used compression rate is 128 kBit/s for MP3.

Analog Dolby Surround Pro Logic

This system consists of the channels front right, front center (center speaker for voices), front left, and rear left, rear right as one channel. The surround channel is applied to the two rear speakers and is time-delayed with regard to the front channels and band-limited to 100 to 7,000 Hz. The center speaker positions the voices to the center, which, in a car, makes localization rather independent of the listening position.

Analog Dolby Surround Pro Logic II

The Dolby Surround Pro Logic II enhances the Pro Logic to a 5.1 system by incorporating two separate surround channels without band limitation, and a subwoofer channel.

Dolby Digital, AC3

AC3 comprises five separate channels without band limitation and a separate subwoofer channel (Low Frequency Effect, LFE) limited to 120 Hz. At a sampling frequency of 48 kHz and a 16-bit resolution, transmission rates of 384 to 448 kbit/s accrue for the data which have a lossy compression ratio of 10:1. For two-channel stereo, the data rate is reduced to 192 kBit/s. Whereas MPEG-2 is the DVD standard in the video field, AC3 is the DVD standard in the audio field. MPEG-2 can only be used on a DVD as an additional audio coding format.

MPEG-2

MPEG-2 is a coding standard which can also be found on a DVD for audio signals, in addition to AC3. The data rate for six channels is about 400 kbit/s. MPEG-2 AAC (Advanced Audio Coding) is an enhancement which manages with half the data rate for the same quality as MPEG-2 and admits the coding of up to 48 audio channels, 16 subwoofer channels and 16 commentary channel. MPEG-2 is used as a standard in digital TV d(DVD satellite, DVD cable, DVD terrestrial). In Japan, MPEG-2 has been used as the exclusive sound format for all digital broadcasting systems since the beginning of the year 2000.

Digital Theatre System (DTS)

DTS is also a 5.1 system. Compared to AC3 it is distinguished by a better sound quality and increased dynamics. DTS has considerably less data compression and works with 1.536 Mbit/s for surround coding.

Logic 7

The 7.1 surround system developed by Lexicon controls the eight independent channels either digitally or develops a corresponding control from a stereo signal on the basis of a matrix decoding. It is thus possible to simulate the surround impression and achieve a very good ambient sound reproduction. A 5.1 source signal is enhanced to 7.1. Logic 7 is often used as professional system in vehicles.

18.5.2 Video Streaming

In the video field, a variety of different formats is applied, driven by different companies and different markets. Figure 18.9 gives an overview of the common graphic/video standards.

Standard	Resolution	Frame / Hz	MPixel/s	RGB 24 (Mb/s)	YUV4:2:0 (Mb/s)
QVGA	320 x 240	60p	4.6	110.6	55.3
NTSC	640 x 480	60i	9.2	221.2	110.6
PAL	720 x 576	50i	10.4	249.8	124.4
VGA	640 x 480	60p	18.4	442.4	221.2
XGA	1024 x 768	60p	47.2	1132.5	566.2
HD Ready	1280 x 720	30p	27.6	663.6	331.8
HD Ready	1280 x 720	60p	55.3	1327.1	663.6
Full HD	1920 x 1080	24p	49.8	1194.4	597.2
Full HD	1920 x 1080	60i	62.2	1493.0	746.5
Full HD	1920 x 1080	60p	124.4	2986.0	1493.0
Full HD 3D	1920 x 2205	24p	101.6	2438.6	1219.3

Graphic standard	Video standard

Fig. 18.9: Overview of common graphic/video standards

Rotating media like video CD, DVD, BD, digital video streams from DVB or DMB tuners, or any kind of portable device will contain compressed video.

Usually, consumer video applications depend on digital data in a compressed format. The reason is clear: it is the cost for either the data transport channel (over the air or the Internet) or the storage capacity (disk space, memory).

For pre-recorded content, there is no need to transport uncompressed video. For compressed video transport, two different principle scenarios are possible. Each scenario has advantages and disadvantages, and these must be weighted under the specific project requirements, including aspects of multi-seat capabilities, function related costs, global costs, flexibility, expandability, bandwidth and other hardware requirements.

Scenario 1: Transcoding at the source

Stream navigation and decoding is done directly at the source. Video and audio are put onto MOST separately from each other. Video is transcoded or re-encoded into a common, to be defined MOST format, e.g., H.264 or MPEG2/4. Audio is transcoded to PCM. Due to the separation of audio and video, the video sink and the audio sink can be independent from each other (example: video is sent to one or more displays, audio is sent to the amplifier). The system uses a push mode from the source towards the sinks. The synchronous nature of MOST with its determined, minimum and constant delay supports the separate transmission of the Audio and Video stream in an optimum way.

Also for DVD (and potentially DivX also) the handling of subpicture information must be considered. It can either be imprinted on the transcoded frame or sent separately (with the video) to the sinks.

Scenario 2: Transfer of program stream and decoding at the video sink

The data is transferred in the original format (Audio + Video, Transport stream or program stream) as provided by the source over MOST to the sink, e.g., a display unit. Decoding and navigation/control are done in the sink.

Audio can be routed back onto MOST towards a detached audio sink (Main Amplifier, detached headphone amplifier). The system may either use a pull mode to request data packets from the source, in case the source is a memory based medium (HDD, Media Drive, …) or a push mode, e.g. for sources like digital tuners. The video sink may then use a push mode to transmit decoded audio and video data to additional sinks which are in copy mode (i.e., viewing/listening to the same content).

18.6 MOST for Driver Assist Applications

Driver assistance systems are about to conquer the market. The latest studies forecast an annual market growth from 9 to 62 million units between 2005 and 2013. In current and future vehicles, driver assistance functions are starting to complete and extend the feature set of traditional infotainment systems. Along with information features such as navigation systems, traffic information, and function warnings, the number of vehicles with driver assistance features like camera systems, distance controls, or lane departure warnings will be rapidly increasing. In this application domain, communication faults become a serious issue since they can lead to severely damage objects or even injure persons. Necessary improvements are being investigated, and recent studies show that MOST might be able to support safety-critical applications as well.

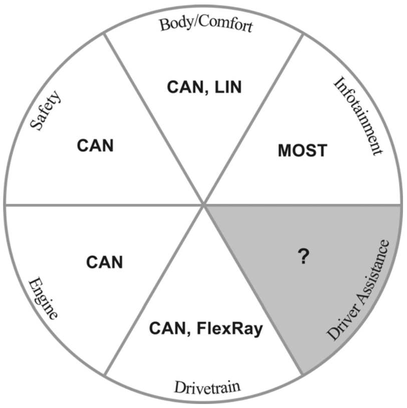


Fig. 18.10: Evolution of Electric/Electronic Domains in the vehicle

Driver assistance systems are on the way to become an integral part of the car, with interfaces to many different clusters of automotive Electric/Electronic systems.

Taking into account the complexity of the applications and the different areas of the car that have to exchange information, it will become obvious that an appropriate network infrastructure is of crucial importance for the efficiency of the system. So in addition to the five classical domains of an electric/electronic network architecture in the car, a new one dedicated to Driver Assist Systems is being introduced (see fig. 18.10).

The existing incarnation of MOST, having been developed and optimized for infotainment, can probably only be used 'as is' for very limited use cases of driver assist applications. On behalf of the MOST Cooperation, investigations are being made on which measures have to be taken in order to fulfill the requirements of the forthcoming safety standard for the automotive industry ISO 26262, which also respects necessary requirements of current standards, e.g., IEC 61508.

With respect to the physical layer, the ring architecture alone has too many limitations for driver assist applications. Therefore, alternative topology options like star or daisy chain are being developed. On the wiring side, fiber optics may not be suitable for all use cases, while at the same time the well established UTP physical layer for MOST50 may either limited in bandwidth expansion or not robust enough for upcoming EMC requirements from hybrid or electric cars. Consequently, the MOST standard is being completed by cost effective, robust alternatives, e.g., Coax cable based physical layers.

As a solution on the protocol side, an architecture with a safety layer as the basis for safety related applications has been proposed. A safety layer provides safe communication over a MOST network using safety codes and a dependable service for the transmission of safety related application data over a MOST network.

Looking into the future, one possible incarnation of the 5th generation MOST systems could be the use in both the infotainment and the driver assist area, supported by different topologies and different physical layers.

Appendix

A The Description Language Message Sequence Chart (MSC)

The automotive area mostly uses MSCs for describing the dynamic behavior of distributed telematics and infotainment systems. The syntax and the semantics of the Message Sequence Charts were standardized in the ITU-T Z.120 [ITUZ120] and in the Corrigendum 1 by the International Telecommunications Union, ITU. A large number of graphic notations for describing the interaction of components were suggested in the past. The Message Sequence Charts proved an intuitive form of describing the asynchronous exchange of messages between the instances of a communication system.

The semi-formal notation originates from the telecommunication area, particularly the areas Requirements Engineering, specification, and the design of telecommunication distribution systems. MSCs were then always considered as one aspect of an SDL (Specification Description Language) model. Meanwhile, however, MSC has detached itself from SDL and the standardization has been pushed on by the ITU.

MSC aims at creating a technical language for specifying and describing the communication behavior of system components among each other and with their environment by means of exchanging messages. As the communication behavior in MSCs is described in an intuitive and transparent manner, the MSC language is easy to learn, use and interpret. In combination with other languages, it supports methods for system specification, system design, simulation, test and documentation.

MSC is independent of the used communication protocol. In addition to the bus communication, the communication models can be specified between the software components within a device, if they are relevant for the system behavior.

The MSC standard supports a modular structure. Recurring communication procedures only have to be described once and can then be used as often as desired as reference. There is no obligatory completeness (in contrast to statecharts) and it is possible to itemize, depending on requirements. Difficult and unclear items are specified in more detail.

However, when describing MOST sequences, a necessary extension for complex data types became obvious. For that reason the so-called *Named Parameters* were introduced, which are described in the MOST MSC Cookbook Specification, chapter 5, appendix A: Extension to MSC2000 (Named Parameters).

MSC defines the two categories: Basic MSCs and HMSCs (High Level MSCs). Figure A.1 illustrates an example of a Basic MSC, referred to as *TuneNextFmStation*. It describes the sequence of changing from a currently selected FM radio station (BBC) to the FM station (RadioTokyo) stored next in the list. The Basic MSC abstracts the sequence and only shows the exchange of information between the main components AmFmTuner_Controller and AmFmTuner. At the start, the system is in its initial state – radio station BBC is selected. The HMI sends the message USERDEF_HMI_Event with the parameter *User selects new Station in ListFM* to the AmFmTuner_Controller. The latter sends the message ListFM.Set with the respective parameters to the AmFmTuner, which executes the action. If the new radio station (RadioTokyo) is tuned, the AmFmTuner sends the new list status (ListFM.Status) and the station name (StationInfo.Status) received via RDS to the AmFmTuner_Controller.

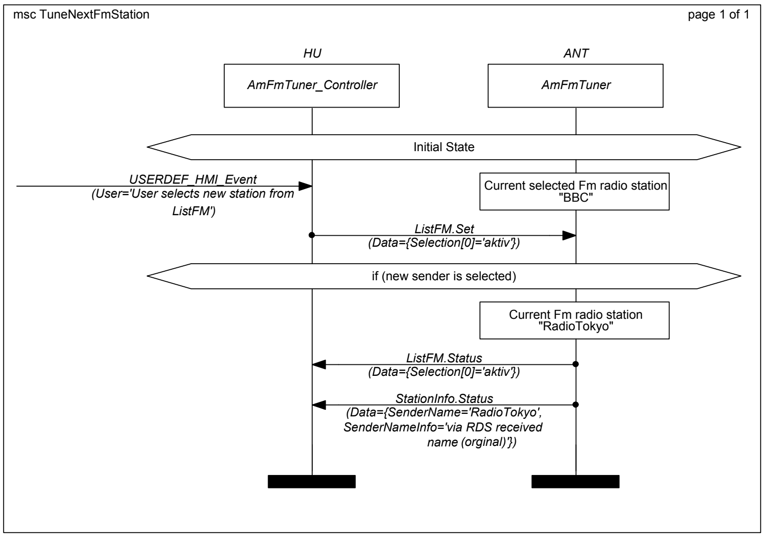

msc TuneNextFmStation
page 1 of 1
HU
ANT
AmFmTuner_Controller
AmFmTuner
Initial State
USERDEF_HMI_Event (User='User selects new station from ListFM')
Current selected Fm radio station "BBC"
ListFM.Set (Data={Selection[0]='aktiv'})
if (new sender is selected)
Current Fm radio station "RadioTokyo"
ListFM.Status (Data={Selection[0]='aktiv'})
StationInfo.Status (Data={SenderName='RadioTokyo', SenderNameInfo='via RDS received name (orginal)'})

Fig. A.1: Example of a Basic MSC

Instances

Instances are, apart from the messages, the most important language constructs of Basic MSCs. Instances are components which exchange messages among each other or with the system environment. CAN nodes, MOST FBlocks etc. are examples of instances. Instances are illustrated as instance head attached to a vertical line. The instance head contains the instance name. Admissible instance names in the MOST domain are, for example, NetBlock, AmFmTuner and PowerMaster. Additionally, the type of instance can be indicated above the instance head (cf. fig. A.1). The end of the instance is described by an instance end symbol (▬). That does not necessarily mean that the instance is completed here. It only means that within this MSC no further events are described for this instance.

MOST uses MOST Instances, as they are called, *General Instances* and *User-Defined Instances*. MOST Instances have a function block name (FBlockID) and an optional instance ID (InstID). The General Instances can be defined freely and are used for instances which cannot be illustrated by a MOST function block.

A user-defined instance is the user interface of a MOST instance, such as the Human Machine Interface (HMI). An abstraction layer separates the user activities in the head unit from the user-defined instance. It translates, for example, the actuation of a pushbutton or switch into events which the user-defined instance can trigger or receive. When the system is modeled, only one single event is thus necessary for the HMI instance for selecting the next radio station stored in the list (TuneNextFmStation, cf. fig. A.1), irrespective of the complexity of the menu navigation.

Messages

MOST Messages in MSCs should always correspond to the MOST function catalog. Messages are illustrated by arrows indicating the time lapse. The arrow end describes the sending and the arrow tip the processing of a message. The labeling consists of the message name and optional parameters which are indicated in brackets underneath the arrow. Messages are arranged along the instance axis according to their chronological appearance. Since the order of sequence is firmly defined, this arrangement is also referred to as *total order*.

Prefixes were introduced to facilitate the differentiation between MOST, CAN and HMI messages. Messages sent by an HMI Instance get the prefix *HMI_Event* and messages received by the HMI Instance get the prefix *HMI_Update*.

The messages *Processing* or *ProcessingAck* can occur in a MOST system if a method was triggered by means of the OPType *StartResult* or *StartResultAck*. If the called instance cannot answer within the predetermined period of time, it sends the message *Processing*, in order to inform the sender that its query is still being processed. Processing messages can occur at any time in any part of an MSC, after the method has been queried. For that reason it does not always make sense to integrate these messages into the MSC, as this makes reading the MSCs difficult.

Environment

The diagram area of an MSC is confined by a rectangular frame, referred to as Environment. It defines the system environment of the MSC. The messages coming from the system environment or being sent there, start and end on the environment (cf. the message USERDEF_HMI_Event in fig. A.1). In contrast to the total order along the instance axes, there is no order defined for events on the Environment.

Actions

In addition to the messages, actions can be specified by instances. An action in an MSC is presented by a rectangular symbol, which can contain any text (cf. fig. A.1 – “currently selected fm station BBC”).

Timer

MSC offers the language constructs *Timer-Start*, *Timeout* and *Timer-Stop* for labeling Timers. Timer-Start defines the starting, Timeout the lapse and Timer-Stop the reset of a Timer. A Timer is always assigned to an instance. For that reason the graphical Timer symbol (cf. fig. A.2) is always connected to the corresponding instance. A Timer symbol includes the name of the Timer and optional Timer parameters. An hourglass symbolizes the Timer-Start, an arrow starting at the hourglass symbolizes the Timeout, and a cross symbolizes the Timer-Stop.

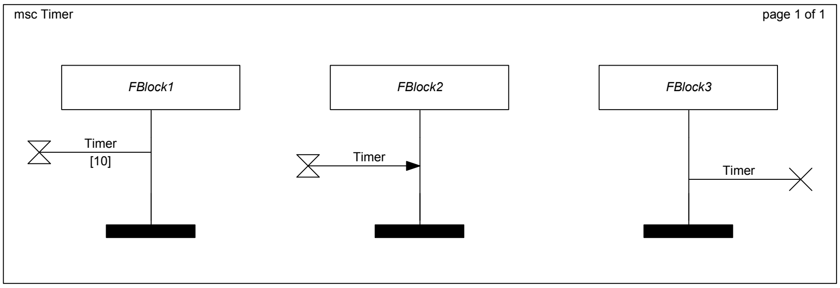


Fig. A.2: MSC Timer symbols

Condition

A condition relates to a number of instances of an MSC. Conditions are graphically represented by hexagons covering the instances they are relating to. In MSCs, conditions are used for describing important system states (e.g., initial state in fig. A.1).

Apart from the Basic MSCs, there are structural language constructs, as they are called, which exceed the description of the message sequence. By means of these language elements, MSCs and parts of MSCs can be combined to complex sequences (*Inline-Expression* and *High-Level MSC*), MSC diagrams can be reused in other MSC diagrams (*References*), MSC instances can be refined (*Decomposition*), and general event structures for instances can be defined (*Coregion and General-Ordering*). However, this clause will only describe High-Level MSC (HMSC). A detailed description of the other language constructs is given in the MSC Specifications ([Z.120 99], [Z.120AB 98], [Z.120C1 01]) and the MOST MSC Cookbook [MSC].

High-Level MSC

Whereas the Basic MSCs describe the communication between different participants on the message level, the High-Level MSCs (HMSC) offer a graphical way of combining individual MSCs with each other (see fig. A.3).

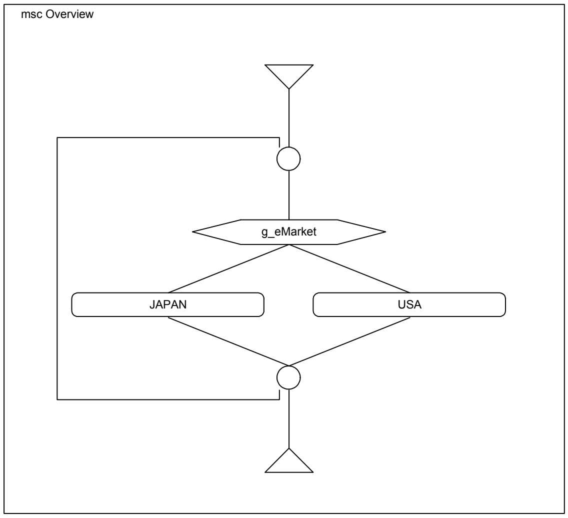


Fig. A.3: Example of a High-Level MSC [MSC]

An HMSC abstracts instances and the message sequence. It is designed for presenting the combination of MSCs. Diagrams of this kind are thus often referred to as *Roadmaps*. By means of the HMSCs, it is possible to present parallel, sequential and alternative combinations of MSCs in a simple manner. The HMSC illustrated in figure A.3 describes the sequence determining the features of a MOST infotainment system depending on the sales market, i.e., the global enum-constant g_eMarket, JAPAN or USA.

This chapter could only deal with the most important elements of the MSC language. Apart from the mentioned constructs, further concepts are specified, making MSC a complete specification language. More information and a complete description of the MSC language, as well as MOST-specific adaptations can be gathered from the SDL forum (http://www.sdl-forum.org) and the MOST Cooperation server (http://www.mostcooperation.com) in [MSC].

B Data rate of the Packet Data Channel of MOST25

The net data rate of the Packet Data Channel of MOST25 basically depends on the bandwidth of the channel and the length of the telegram. The number of telegram segments is calculated according to the following equation [DS 8104]:

$$\begin{aligned} Tp &= roundup\left(\frac{Pd + Ph}{Af}\right) \\ &= roundup\left(\frac{Pd + Ph}{(15 - SBC) * 4}\right) \qquad \text{(Equation B.1)} \\ &= roundup\left(\frac{Pd + 10}{(15 - SBC) * 4}\right) Frames \end{aligned}$$

The transmission time of a telegram depends on the sample rate (number of frames per second) and a delay time, which is required by the MOST Network Interface Controller for providing the next telegram:

$$Tt = \frac{Tp + Ta}{Fs} s \qquad \text{(Equation B.2)}$$

The net data rate is calculated with Tt and Pd according to the equation:

$$R = \frac{Pd * 8}{Tt} Bit / s \qquad \text{(Equation B.3)}$$

with:

Tp:	Number of telegram segments
Pd:	Number of data bytes in a telegram
Ph:	Number of control bytes in a telegram

Arbitration	1 Byte
Target address	2 Bytes
Data area length	1 Byte

Source address	2 Bytes
CRC	4 Bytes
Total	10 Bytes

Af:	Bandwidth of the Packet Data Channel
Tf:	Transmission time of a telegram
Ta:	Interval between two telegrams of a node; indicated in number of the frames (always 4 frames in the case of the NIC)
SBC:	Value of the synchronous bandwidth control register (between 6 and 15)
Fs:	Sample rate (typically 44.1 kHz)
R:	Net data rate

Table B.1 summarizes the net data rates for typical telegram lengths and possible bandwidths of the packet data:

SBC	*Pd*						*Unit*
	1014	*512*	*256*	*128*	*64*	*48*	
6	10.841	9.507	7.526	5.645	3.226	2.822	Mbits/s
7	9.937	8.602	6.947	5.018	3.226	2.822	Mbits/s
8	8.725	7.854	6.451	5.018	3.226	2.419	Mbits/s
9	7.611	6.947	5.645	4.516	2.822	2.419	Mbits/s
10	6.388	5.827	5.018	4.105	2.822	2.419	Mbits/s
11	5.261	4.882	4.301	3.474	2.509	2.117	Mbits/s
12	3.975	3.763	3.345	2.822	2.053	1.882	Mbits/s
13	2.710	2.580	2.377	2.053	1.613	1.411	Mbits/s
14	1.376	1.338	1.272	1.158	0.982	0.891	Mbits/s
15	0	0	0	0	0	0	Mbits/s

Table B.1: Net bit rate of the packet data with Ta=4 and Fs=44.1 kHz

Note: For the data rate of the Packet Data Channel of MOST50 and MOST150, see sections 5.1 and 5.6.

C Abbreviations

8B10B	8B/10B Code
A/V	Audio/Video
A2DP	Advanced Audio Distribution Profile - a Bluetooth profile
AAC	Advanced Audio Coding
ABY	All-Bypass
ACL	Asynchronous Connection-Less - an asynchronous Bluetooth link
ADS	Asynchronous Data Service
AG	Audio Gateway - Bluetooth
AH	Address Handler
AM	Amplitude Modulation
AMA	Active Member Address
AMS	Application Message Service
ANT	Antenna Tuner
API	Application Programming Interface
APWD	Average Pulse Width Distortion
ASYNC	Parallel Asynchronous Mode of the SP; pin of the NIC
AV/C	Audio-Video Compatibility Specifications
AVDTP	Audio/Video Distribution Transport Protocol - a Bluetooth profile
BAP	Operation and Display Protocol (German: Bedien- und Anzeigeprotokoll)
BOSS	Bus-Owner/Supervisor/Selector
BPF	Bits-per-Frame
PACK	preemptive acknowledge
CACK	complete acknowledge
CAI	Cavity-as-Interface
CAL	CAN Application Layer
CAN	Controller Area Network
CAPL	Communication Access Programming Language; programming language similar to C by Vector Informatik
CDEF	CAN Data Exchange Format
CI	Configuration Interface
CM	ConnectionMaster (also: Connection Manager)
CMD	Command Interpreter
CMS	Control Message Service
CP	Control Port of the NIC
CSMA	Carrier Sense Multiple Access

CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access/Collision Detection
CSR	Control and Status Register
CU	Critical Unlock
DAP	Diagnostic Adaptation Protocol
DDJ	Data-Dependent Jitter
DHWG	Digital Home Working Group
DoC	Declaration of Compliance
DSD	Direct Stream Digital
DSP	Digital Signal Processor
DSP	Digital Sound Processor
DTC	Diagnostic Trouble Code
DTD	Document Type Definition
DTS	Digital Theatre System
ECL	Electrical Control Line
EHC	External Host Controller
EHCI-State	External Host Controller Interface State Machine
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
EOC	Electrical/Optical Converter
EOP	End of Production
ePhy	electrical Physical Layer
ET	FBlock EnhancedTestability
FBlock	Function Block
FBlockID	Function Block Identifier
FCS	Frame Check Sequence
FHSS	Frequency Hopping Spread Spectrum (Bluetooth)
FIFO	First-In First-Out Buffer
FktID	Function Identfier
FM	Frequency Modulation
FOR	Fiber Optic Receiver
FOT	Fiber Optic Transceiver
FOX	Fiber Optic Transmitter
FPGA	Field-programmable Gate Array
FS	Frame Synchronization; sample rate
FSY	Synchronization Pin of the INIC
FTDMA	Flexible Time Division Multiple Access
FWHM	Full-Width at Half-Maximum
GAVDP	Generic Audio/Video Distribution Profile - a Bluetooth profile
GMI	Golden MOST Implementations
GOEP	Generic Object Exchange Profile - a Bluetooth profile
GSM	Global System for Mobile Communications - a Bluetooth profile

GW	Gateway
HAVi	Home Audio-Video Interoperability
HCI	Host Controller Interface - a Bluetooth interface
HF	Hands-Free Unit - a Bluetooth interface
HMI	Human Machine Interface
HU	Head Unit
IEEE	Institute of Electrical and Electronics Engineers
IFG	Inter Frame Gap
INIC	Intelligent Network Interface Controller
InstID	Instance ID of a function block
IP	Intellectual Property
ISM	Industrial Scientific Medical Band; unlicensed frequency band
ITU	International Telecommunications Union
JPEG	Joint Photographic Experts Group
JTAG	Joint Test Action Group
L2CAP	Logical Link Control and Adaption Protocol - a Bluetooth protocol
LAN	Local Area Networks
LED	Light Emitting Diode
LEG	LEG OS8104 compatibility mode; pin of the NIC OS8104A
LFE	Low Frequency Effect
LLC	Logical Link Control
LLD	Low Level Driver
LMP	Link Manager Protocol - a Bluetooth protocol
MAC	Media Access Control
MAMAC	MOST Asynchronous Medium Access Control
MBM	Message Buffer Management
MCA	MOST Compliance Administrator
MCPL	MOST Compliance Product List
mCRA	Channel Resource Allocation Table
MCS	MOST Transceiver Control Service
MCSB	MOST Compliance Supervisory Board
MCTG	MOST Compliance Technical Group
MCTH	MOST Compliance Test House
MDM	MOST Debug Message Module
MDP	MOST Data Packets
MEP	MOST Ethernet Packets
MediaLB	Media Local Bus
MHP	MOST High Protocol
MIMO	Multiple Input-Multiple Output
MIS	Message Interface Service
MLBCLK	MediaLB Clock
MLBDAT	MediaLB Data

MLBSIG	MediaLB Signal
MNS	MOST NetServices Kernel
MPEG	Moving Picture Expert Group
MPR	Maximum Position Register
MRT	MOST Routing Table
MSC	Message Sequence Chart(s)
MSV	MOST Supervisor
Mutex	Mutual Exclusion
NA	Numerical Aperture
NAV	Navigation Device
NB	NetBlock
NBEHC	NetBlock on EHC (cooperates with NBMIN)
NBMIN	Minimum NetBlock on INIC
NCE	Network Change Event
NDIS	Network Driver Interface Specification
NIC	Network Interface Controller
NTFS	Notification Service
OEC	Optical/Electrical Converter
OEM	Original Equipment Manufacturer
OpCodes	Operation Codes
oPhy	optical Physical Layer
OpType	Operation Type
OS	Operation System
OSI	Open Systems Interconnection Reference Model
OUI	Organizationally Unique Identifier
PAL	Protocol Adaption Layer
PAR_CP	Control Port Parallel; pin of the NIC
PAR_SRC	Source Port Parallel; pin of the NIC
PARC	Xerox Palo Alto Research Center
PC	Physical Channel
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PCM	Pulse Code Modulation
PCS	Plastic Clad Silica
PDC	Packet Data Channel
PIM	Personal Information Management
PLL	Phase-locked Loop
PM	Port Message Protocol
PMA	Parked Member Address
PMMA	Polymethyl Methacrylate
PMS	Port Message Service
POF	Plastic Optical Fiber
ppm	parts per million

PWD	Pulse Width Distortion
PWV	Pulse Width Variation
PXI	PCI eXtensions for Instrumentation; modular instrumentation platform originally introduced in 1997 by National Instruments
QA	Quality Assurance
QoS	Quality of Service
RCLED	Resonant Cavity Light Emitting Diode
RBD	Ring Break Diagnostic
RFCOMM	Radio Frequency Communication - a Bluetooth protocol
RMCK	Recovered Master Clock of the NIC or INIC
RMS	Root Mean Square
RTCP	Real Time Control Protocol
RTP	Real Time Transport Protocol
RTR	Remote Transmission Request
SA	Super Audio by Sony
SCK	Clock Pin
SCL	Clock of the serial CP Interface; pin of the NIC
SCO	Synchronous Connection Oriented – a Bluetooth link
SCS	Synchronous Connection Service
SDP	Service Discovery Protocol - a Bluetooth protocol
SH	Socket Handle
SI	Step Index
SIG	Special Interest Group
SIM	Subscriber Identity Module
SNAP	Sub Network Access Protocol
SOP	Start of Production
SP	Source Port of the NIC
SR	Streaming Data Receiver
SSO	Sudden Signal Off
STP	Shielded Twisted Pair
SX	Streaming Data Transmitter
SyncML	Synchronization Markup Language
SW	Software
TCU	Telephone Control Unit
TDM	Time Division Multiplexing
TDMA	Time Division Multiple Access
TP2.0	proprietary Volkswagen Transport Protocol
TSI	Transport Stream Interface
TV	Television Tuner
UDS	Unified Diagnostic Services
UI	Unit Interval; shortest pulse in the data signal
UJ	Uncorrelated Jitter
UPnP	Universal Plug-and-Play

USB	Universal Serial Bus
UTP	Unshielded Twisted Pair
vMSV	Virtual MOST Supervisor
wADS	Asynchronous Data Service Wrapper
wAMS	Application Message Service Wrapper
wCMS	Control Message Service Wrapper
WLAN	Wireless Local Area Network
wMCS	MOST Processor Control Service Wrapper
wSCM	Socket Connection Manager Wrapper
www	World Wide Web

D Glossary

API
The application programming interface is an interface which allows other programs to access a software element (e.g., communication interface).

Attenuation
The positive (decrease) or negative (increase) attenuation of a signal is usually given in dB (decibel).

Biphase Mark
Biphase mark code refers to a line code that differs slightly from the Manchester code. There is always a transition at the start of a bit. A 1 has a transition at the middle of the bit. For a 0, there is no transition until the end of the bit.

Bond Breakage
Bond breakage is an interruption of the wires that connect a chip electrically to the lead frame.

Boundary Descriptor
The boundary descriptor determines the rate of streaming and packet data within a frame.

Bypass
The bypass is a function in the MOST Network Interface Controller. It is closed during the initializing phase, in the case of a device error or when the device is deliberately switched off. It transmits all data arriving at the input (RX) directly to the output (TX).

Cavity
A cavity is a hollow space in the pigtail.

Central Registry
The Central Registry is a table containing the function blocks existing in the system, as well as the logical addresses of the devices on which the function blocks are implemented. It is implemented on the NetworkMaster.

Cladding
Cladding is the optical coat of the POF.

Coating
Coating is the exterior coating of the POF. Vehicle applications use a double exterior coating.

Codec
Codec is a portmanteau word formed from the words coder and decoder. It refers to a process or program for digitally coding and decoding data or signals.

ConnectionMaster
The function block ConnectionMaster manages the streaming channels in a MOST system.

Control Command
Control commands are data packets for controlling the MOST system. They are transmitted mostly on the Control Channel, but can also use the Packt Data Channel.

Controller
A Controller controls function blocks. It is often responsible for a specific function in the MOST system (e.g., audio function).

Decentral Registry
The Decentral Registry is an image of the Central Registry (or of a section thereof) in a NetworkSlave, which accesses other function blocks in a MOST system.

Delamination
The disconnection of pottant and lead frame of an IC is referred to as delamination. It is a characteristic damage after cyclic thermal stress.

Deployment
Deployment is the distribution of functions to the actual hardware structure of the on-board electrical infrastructure.

Device
A MOST device is a physical device that is connected to the MOST system via a Network Interface Controller.

DeviceID
The DeviceID stands for a physical device or a group of devices in the network. The DeviceID (RxTxAdr) can represent a node position address (RxTxPos), a logical address (RxTxLog) or a group address.

Errata
Errata is the complementary document of a specification in which errors are corrected or matters are clarified. In MOST specifications, it is used in compliance specifications.

Extrusion
Extrusion (Latin: extrudere = to eject, to extrude) refers to a continuous process in which plastics are pressed through a nozzle.

f(x) = f(f(x)), i.e., if the repeated application of a function is equivalent to the single application.

FBlock Shadow
The FBlock shadow is an implementation concept in which the FBlock is controlled via a proxy.

Ferrule
A ferrule is a guide tube which accommodates the POF fiber and forms the mechanical interface for fixing the fiber.

Flash Memory
Erasable/re-writable memory; in the automotive area the software of the micro controllers is usually filed in flash memories.

Frame
A frame is the basic data structure of the MOST data link layer which is transmitted cyclically in the MOST ring. A frame carries data belonging to the Streaming Channel, Packet Data Channel and the Control Channel.

Fresnel Losses
Losses occurring at the radiative transition between media of differing refractive indices are referred to as Fresnel losses.

Function Block or FBlock
A function block or FBlock is an object containing the interface, i.e., the methods and properties, for controlling a specific application (e.g., a radio tuner or a telephone interface) or system service.

Function
Function is the generic term for the properties and methods of a function block.

Gateway
A gateway connects different data transmission systems. In this connection the complete communication stack of each system must be implemented (all existing layers of the OSI model).

Golden Fiber
Golden fiber is the term for a reference standard (reference fiber).

Hygroscopic Material
Hygroscopic material is a material that absorbs moisture. If mishandled, it can cause problems during the welding process.

Idempotent
Idempotent refers to the property of a function or a method if the following applies:

Init Ready
The Init Ready event informs the application of the MOST system having changed into NetInterface Normal Operation.

Leadframe
A thin layer of metal that has pre-stamped connecting pins and bond pads of integrated circuits (ICs) is referred to as a leadframe.

Light-off State
The light-off state refers to the state in which the modulated signal is switched off by any node. All nodes then change to the light-off state, pass to the NotOK system state and and finally end in the NetInterface Off state.

Light-on State
Light-on describes the state in which the modulated signal is switched on.

Lock Event
The lock event is generated when the MOST Network Interface Controller could synchronize steadily onto the signal for the first time or after an unlock event.

Lock
Lock refers to the TimingSlaves synchronizing onto the MOST signal.

Method
A method is a function of a function block by means of which an action is started, which then leads to a result after a specified period of time.

Mode Dispersion
Mode dispersion is the pulse broadening in optical fibers due to differing propagation constants of the modes or light beams.

NetBlock
NetBlock is a function block that is implemented in each device for administering functions.

NetInterface
The NetInterface comprises the entire MOST interface consisting of the physical layer, MOST Network Interface Controller and Network Service.

NetServices
See Network Service

Network Change Event (NCE)
A network change event occurs if the number of active nodes in the ring changes, e.g., if during the start-up a device needs more time for the boot-up process and closes its bypass late or if a node fails or changes into sleep mode.

Network Scan
See: system scan

Network Service
The Network Service is the driving layer that must be implemented in each device of a MOST system for controlling the MOST Network Interface Controller. Applications and system services access the MOST system via the Network Service. NetServices is the trade name for the implementation of the SMSC Network Service.

NetworkMaster
The function block NetworkMaster controls the system state and manages the Central Registry.

Numerical Aperture
The numerical aperture describes the aperture angle in optical fiber technology, under which optical radiation can be coupled into an optical fiber.

Operation
The MOST standard defines specific standard operations that can be used for properties and methods.

Packet Data
Packet data are large data packets that are transmitted on the Packet Data Channel, e.g., when tunneling a TCP/IP connection via the MOST bus.

Pigtail
A two-fiber cable with a POF connector pre-assembled at one end, consisting of an optical transmitter, an optical receiver, two POF fibers and the mechanical adaptation is referred to as a pigtail.

PLL (phase locked loop)
A phase locked loop is a control circuit that tries to keep an oscillator in the phase of the input frequency by means of phase detection.

POF (polymeric optical fiber, plastic optical fiber)
A polymeric optical fiber is a plastic optical fiber for transmitting data. The core of the POF is mostly made of polymethyl methacrylate (PMMA). The optical fiber is often coated in order to achieve a higher stability.

PowerMaster
The PowerMaster is responsible for booting up and shutting down a MOST system and supervises its power management.

Property
A property is a function that corresponds to a specific property of a function block. The properties serve to query or change the current state of a function block

Quadlet
A quadlet consists of 4 bytes.

Recombination
Recombination refers to the combination of positive and negative charge carriers (ions, electrons) to form an electrically neutral product (atom, molecule).

RxTxAdr
Generic term for the three address types of a MOST node: logical node address (RxTxLog), node position address (RxTxPos) or group address. The transmitter address is TxAdr and the receiver address RxAdr.

RxTxLog
RxTxLog is the logical address. TxLog refers to the logical transmitter address and RxLog to the receiver address. It may only appear once in the MOST ring.

RxTxPos
RxTxPos describes the node position address of a node. It depends on the position of the node in the ring. TxPos refers to the transmitter and RxPos to the receiver address.

S/PDIF
S/PDIF stands for Sony/Philips Digital Interface. It is a bus and interface specification for transmitting digital audio signals between different devices and is also referred to as Sony/Philips Digiconnect Format, S/P-DIF or TOSLINK.

Slave
A slave is a function block that is controlled by a Controller, or the device in which it is implemented.

SPY Mode
The SPY mode is an operational mode for analysis tools. They are not visible in the network but can listen into the entire bus communication.

Streaming Data
Streaming data can be audio or video data, for example. In MOST, they are transmitted on the Streaming Data Channel.

Substitution Method
The substitution method is a comparative measuring method, in which the characteristics of a fiber under test are compared to the respective characteristics of a reference fiber.

System Scan
System scan describes how the function block configuration data of the Network-Slaves are read by the NetworkMaster. This is also often referred to as network scan.

System State
During normal operation the system is in the OK state, during the initialization or in the case of an address conflict the system is in the NotOK state.

t_{Diag_Signal}
Time a TimingSlave node waits for activity at its Rx input before switching to ring break TimingMaster mode.

TimingMaster
The TimingMaster generates the system clock for the frames and blocks. The TimingSlaves synchronize onto the system clock.

Transceiver
Transceiver is a portmanteau word formed from the words transmitter and receiver.

$t_{SSO\_ShutDown}$
Time between setting the Shutdown Flag and switching off the signal at the output.

$U_{Critical}$
Critical voltage $U_{Critical}$ is the limit at which the application might no longer work safely but where communication is still possible. In that case, a source must route zeros and a sink must secure its output signals. The Mute property remains unchanged. In case of a recovery, the output signals can be restored immediately.

U_{Low}
Low voltage U_{Low} is a device specific limit, where even the NetInterface no longer works reliably, so even communication cannot be maintained. If U_{Low} is reached, the device switches off the modulated signal and switches to DevicePowerOff Mode. The device stays in DevicePowerOff mode, even if the supply voltage recovers.

Unlock Event
The unlock event is generated for the application by the Network Service if the MOST Network Interface Controller can no longer generate a stable clock from the PLL.

Unlock
A TimingSlave that can no longer synchronize with its PLL to the input signal at the Rx input detects an unlock.

Wave Soldering
Wave soldering is a processing procedure in electronics production. Printed circuit boards are passed across a wave of molten solder, creating a reliable connection between the PCB and the component.

E Bibliography

[AppN Mig]	Application Note MOST150 Migration, Rev 1.0; 05/2009; MOST Cooperation
[Bluetooth]	Specification of the Bluetooth System-Profile, Version 1.1, 02/2001; Special Interested Group Bluetooth
[Burton 2004]	Automotive Testing of Automotive Telematics Systems using TTCN-3, Simon Burton, Andre Baresel, Ina Schieferdecker, Proceedings of the 3rd Workshop on System Testing and Validation (SV04), December 2004.
[Cat 02]	MOST Function Catalog Read Me First; Rev 1.0; 12/2002; MOST Cooperation
[Com_ePhyL]	MOST ePHY Compliance Test Spec (with ePhy Compliance Test Patterns) Rev 1.0; 06/2006; MOST Cooperation
[ComPhyL 1.0]	MOST Specification MOST Compliance Test of Physical Layer; Rev 1.0; 12/2003; MOST Cooperation
[ComPhyL 150]	MOST150 oPhy Compliance Measurement Guideline; Rev. 1.1; 08/2010; MOST Cooperation
[ComPhyL_Err]	ERRATA MOST Compliance Test of Physical Layer; Rev 1.0, 05/ 2006; MOST Cooperation
[ComPhyLP 1.0]	MOST Compliance Verification Procedure Physical Layer; Rev. 1.0; 07/2004; MOST Cooperation
[ComPhyLP 150]	MOST150 oPhy Compliance Verification Procedure – Physical Layer; Rev. 1.1; 07/2010; MOST Cooperation
[ComPhyLP_Err]	ERRATA MOST Compliance Verification Procedure Physical Layer; Rev. 1.1; Rev. 1.0; 07/2005; MOST Cooperation
[ComProf]	MOST Profile Compliance Test Specification; Rev 1.0; 10/2005; MOST Cooperation
[ComReq]	MOST Specification - MOST Compliance Requirements, Rev 2.1; 04/2009; MOST Cooperation
[ContProt]	MOST Content Protection Scheme DTCP Implementation; Rev. 3.0; 08/2009; MOST Cooperation

[Core]	MOST Core Compliance Test Specification; Rev 1.1.01; 06/2005; MOST Cooperation
[Core_Err]	ERRATA MOST Core Compliance Test Specification; Rev 1.3; 07/2009; MOST Cooperation
[Dau 01]	Daum W. et al.: „POF – optische Polymerfasern für die Datenkommunikation", Springer Verlag, 2001
[DPA]	Diagnostic Protocols Adaptation Specification, Rev. 1.0.1; 08/2009; MOST Cooperation
[DS 8104]	OS8104 – MOST Network Transceiver – Final Product Data Sheet; Jan. 2003; www.smsc-ais.com
[DS FCM110]	MOST Fiber Optic Transceiver, FCM110R/ FCM110D, Fire-comms Ltd.
[DTD Cook]	MOST DTD Cookbook; Rev 1.2; 12/2009; MOST Cooperation
[DynSpec]	MOST Dynamic Specification; Rev 3.0; 07/2008; MOST Co-operation
[ECL]	Electrical Control Line, Rev. 1.0.1; 08/2009; MOST Coopera-tion
[ETS 300 406]	ETS 300 406 „Methods for Testing and Specifications (MTS); Protocol and Profile Conformance Testing Specifications; Standardization methodology” http://portal.etsi.org/mbs
[ETSI 2003]	ETSI ES 201873: The Testing and Test Control Notation TTCN-3, Part 1 (Core Language), Part 2 (Tabular Format), Part 3 (Graphical Format), Part 4 (Operational Semantics), Part 5 (TTCN-3 Runtime Interface), Part 6 (TTCN-3 Control Inter-faces), Sophia-Antipolis, France, October 2003.
[FBADiskPl 2.4]	MOST FunctionBlock AudioDiskPlayer; Rev 2.4; 09/2003; MOST Cooperation
[FBAmFmT 2.4]	MOST FunctionBlock AmFmTuner, Rev 2.4.2; 09/2003; MOST Cooperation
[FBAudioA 2.4]	MOST FunctionBlock AudioAmplifier; Rev 2.4.2; 09/2003; MOST Cooperation
[FBAuxIn 3.5]	MOST FunctionBlock AuxIn, Rev 3.5; 10/2008; MOST Coop-eration
[FBCM]	MOST FunctionBlock ConnectionMaster; Rev 3.0.1; 09/2010; MOST Cooperation

[FBET]	MOST FunctionBlock EnhancedTestability; Rev 3.0; 8/2010; MOST Cooperation
[FBGenPl 2.5]	MOST FunctionBlock GeneralPlayer; Rev 2.5.1; 01/2008; MOST Cooperation
[FBVehicle 1.6]	MOST FunctionBlock Vehicle, Rev 1.6 09/2002; MOST Cooperation
[FBX Cook]	MOST FIBEX Cookbook; Rev 1.0; 03/2009; MOST Cooperation
[Found 04]	MOST Foundation Training; Training Handout; Oasis SiliconSystems; 2004
[GADV]	4CS, GADV Gmbh, http://www.gadv.de/
[GenFB]	MOST FunctionBlock GeneralFBlock; Rev 3.0.2; 09/2010; MOST Cooperation
[Gus 98]	D. Gustedt, W. Wiesner: „Fiber-Optic-Übertragungstechnik", Sende- und Empfangsgrundlagen, Franzis-Verlag, 1998
[HFP 1.5]	Hands-Free Profile 1.5; HFP1.5_SPEC; 2005-11-25 www.bluetooth.org
[ISO 14229]	Road vehicles -- Unified diagnostic services (UDS), ISO 14229. ISO, Geneve 2009
[ISO 14230]	Road vehicles -- Diagnostic systems -- Keyword Protocol 2000, ISO 14230, Geneve 1999/2000
[ISO 9646]	Information Technology, Open Systems Interconnection, Conformance Testing Methodology and Framework. International Standard IS-9646. ISO, Geneve 1991
[ITUZ120]	Z.120: Message Sequence Charts, MSC-2000, Geneva, Switzerland, October 1999.
[Lehmann 2000]	Test Case Design by Means of the CTE XL. E. Lehmann; Wegener, J Proceedings of the 8th European International Conference on Software Testing, Analysis & Review (EuroSTAR 2000), Kopenhagen, Denmark, December 2000.
[MAMAC]	MAMAC Specification; Rev. 1.1; 12/2003; MOST Cooperation
[MediaLB]	Media Local Bus Specification; Physical Layer and Link Layer; Version 3.0; SMSC; www.smsc-ais.com
[Mercury]	Test Director, Mercury Interactive Limited, http://www.testerlyzer.de/

[MHP] MOST High Protocol Specification; Rev 2.3; 12/2008; MOST Cooperation

[MNS L1 1.x] MOSTNetServices; Basic Services (Layer I); Programmers' Library for Interfacing to MOST User Manual and Specification of Layer I; Preliminary Version 1.10; www.smsc-ais.com

[MNS L1 2.x] MOST NetServices Layer I; Wrapper for INIC; V2.1.x; User Manual/Specification; www.smsc-ais.com

[MNS L1 3.x] MOST NetServices Layer I; Wrapper for INIC; V3.x; User Manual/Specification; www.smsc-ais.com

[MNS L1 Exa] MOST NetServices; Layer 1 Example; Version 1.10-01; www.smsc-ais.com

[MNS L2] MOST NetServices Layer II; User Manual/Specification; Rev. 1.10.x / 2.1.x; www.smsc-ais.com

[MOST 2.4] MOST Specification; Rev 2.4; 05/2005; MOST Cooperation

[MOST 2.5] MOST Specification; Rev 2.5; 10/2006; MOST Cooperation

[MOST 2003] Empfehlungen für Optische MOST Interfaces in Automotive Anwendungen (Version 1.0), MOST Co, Audi AG, BMW AG, DaimlerChrysler AG, RMCTech GmbH, 2003.

[MOST 3.0] MOST Specification; Rev 3.0 E2; 07/2010; MOST Cooperation

[MOST AARSMD] Automotive Application Recommendation for optical MOST Components - Surface Mount Device (SMD) Rev. 1.0, 10/2010, MOST Cooperation

[MOST AARTHM] Automotive Application Recommendation for optical MOST Interfaces, Rev. 3.0-06, 06/2009, MOST Cooperation

[MOST AppTF] Application Note POF Transfer Function, Rev. 1.0, 03/2009, MOST Cooperation

[MOST BaPhy] MOST Physical Layer Basic Specification, Rev. 1.0, 12/2008, MOST Cooperation

[MOST ePhy] MOST Specification of Electrical Physical Layer, Rev. 1.0, 12/2004, MOST Cooperation

[MOST Org] MOST Cooperation Organizational Procedures Rev 3.5 04/2007; MOST Cooperation

[MOST Phy150] MOST150 oPHY Automotive Physical Layer Sub-Specification, Rev. 1.1, 05/2010, MOST Cooperation

[MOST Stream]	Stream Transmission Specification Rev. 3.0. 08/2009, MOST Cooperation
[MSC]	MOST MSC Cookbook; Rev 1.2; 08/2005; MOST Cooperation
[NetBlock]	MOST FunctionBlock NetBlock; Rev 3.0.1; 09/2010; MOST Cooperation
[Optitas]	Tool4M-XL, Optitas GmbH, http://www.optitas.de/
[PBAP]	Phone Book Access Profile; PBAP_SPEC; www.bluetooth.org
[PFLNet]	MOST NetServices API – The Interface to the MOST Network; Product Flyer 04/03; SMSC; www.smsc-ais.com
[Por 2004]	Vernetzte Systeme im Cayenne; Porsche Engineering Magazine; Ausgabe 01/2004; S. 12 - S.15
[Praesideo]	Praesideo data sheet; www.bosch-sicherheitsprodukte.de (Nov. 2010)
[RFC 2616]	Hypertext Transfer Protocol – HTTP/1.1; Juni 1999; R. Fielding, J. Gettys, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee
[Ruetz]	Testerlyzer, Ruetz Technologies GmbH, http://www.testerlyzer.de/
[SAP]	SIM Access Profile; SAP_SPEC; www.bluetooth.org
[Schr 01]	A. Schramm, "MOST Investigations at BMW, Migration or Evolution“, Annual All Members Meeting, Frankfurt, Germany, 11th March 2008
[Schr 02]	A. Schramm, “MOST in der dritten Generation: MOST150 Systemaspekte und Migrationsalternativen“, ElektronikPraxis; 8th August 2008
[SMSC]	Optolyzer G2, SMSC, Automotive Infotainment Systems GmbH, www.smsc-ais.com
[Stream]	MOST Specification for Stream Transmission; Rev. 3.0; 07/2010; MOST Cooperation
[Tutorial]	MOST Tutorial-CD; MOST Cooperation
[Vector]	CANoe- Development and Test Tool for CAN, LIN, MOST, FlexRay, Ethernet and J1708, www.vector.com
[Vog 02]	E. Voges, K. Petermann: „Optische Kommunikationstechnik“, Springer Verlag, 2002

[Wei 99]	Weinert A.: „Plastic Optical Fibers“, Publicis MCD Verlag, Erlangen, 1999
[Z.120 04]	ITU-T Message Sequence Chart (MSC); Geneva; (04/2004); http://www.itu.int
[Z.120 99]	ITU-T Recommendation Z.120 (MSC2000) Message Sequence Chart (MSC); Geneva; (11/1999); http://www.itu.int
[Z.120AB 98]	ITU-T Message Sequence Chart (MSC) Annex B: Formal sematics of Message Sequence Charts; Geneva; (04/1998); http://www.itu.int
[Z.120C1 01]	ITU-T Z.LC-Text: Recommendation Z.120 Corrigendum 1; Geneva; (11/2001); http://www.itu.int

Index

MOST®

THE AUTOMOTIVE MULTIMEDIA NETWORK

MOST (Media Oriented Systems Transport) is a multimedia network technology developed to enable an efficient transport of streaming, packet and control data in an automobile. It is the communication backbone of an infotainment system in a car. MOST can also be used in other product areas such as driver assistance systems and home applications.

MOST is a communication system with a new, flexible architecture, used by many different manufacturers. It is the most widely used multimedia network in the automotive industry.

In 2010, more than 100 car models are already equipped with MOST Technology. MOST defines the protocol, hardware, software and system layers that are necessary for the high-performance and cost effective transport of realtime data in a shared medium. MOST Technology is developed and standardized within the MOST Cooperation through a joint effort of carmakers and their suppliers. It is now evolving into a third generation with MOST150 Technology.

The MOST Cooperation commissioned this updated book with the goal of making it easier to understand and use MOST Technology. The authors of the book work for carmakers, suppliers and scientific institutes. The emphasis of the first part of the book is the MOST standard itself. After describing the organization of the MOST Cooperation, the general communication architecture of an automobile using MOST is presented. Then the chapter "Survery of the System Architecture" gives an overview of MOST Technology. The most important aspects of the technology are then presented in detail in subsequent chapters.

The book is aimed at engineers at suppliers and carmakers that want to get started with MOST Technology. It also provides an overview of an effective communication architecture to interested engineers outside the automotive industry.

EUR 49,95 [D]
ISBN 978-3-645-65061-8

9 783645 650618

Visit our website: www.franzis.de