VOR DEM AMT FÜR PATENTE UND HANDELSMARKEN DER VEREINIGTEN STAATEN

In re *Inter Partes* Begutachtung von:	)
U.S. Patentnr. 7,489,786	)
Erteilt: 10. Februar 2009	)
Antragsnr.: 10/316,961	)
Eingereicht am: 11. Dezember 2002	)

Für: **Audiogerät-Integrationssystem**

EINGEREICHT VIA E2E

STELLUNGNAHME VON DR. WOLFGANG BOTT GEMÄß 37 C.F.R. § 1.68

Ich, Dr. Wolfgang Bott, erkläre hiermit das Folgende:

1. Ich bin gegenwärtig Technischer Koordinator bei der MOST Cooperation mit Sitz in Karlsruhe, Deutschland. Meine beruflichen Zuständigkeiten umfassen die Koordination der Arbeitsgruppen und die Weiterentwicklung der Spezifikation. Als Teil meiner Arbeit bewerbe ich die MOST Spezifikation, ermutige andere in der Branche die MOST Spezifikation einzusetzen und leite andere in der Branche darin an, wo diese zu finden ist.

2. 2003 begann ich meine Arbeit als Berater für die MOST Cooperation und 2005 wurde ich zum Technischen Koordinator der MOST Cooperation ernannt. Angesichts meiner langen Zeit bei der MOST Cooperation, bin ich im Allgemeinen vertraut mit der Geschichte der MOST Cooperation, ihres Geschäftsbetriebs und ihrer Praktiken, sowohl vor als auch nach meinem Eintritt. Ich gebe diese Erklärung auf der Grundlage meines eigenen persönlichen Wissens ab.

3. Die MOST Cooperation besteht aus einem Partnerschaftsnetzwerk aus Autoherstellern, Geräteherstellern, Systemarchitekten und Lieferanten von Schlüsselkomponenten. Die MOST Cooperation ist verantwortlich für die Weiterentwicklung und Bewerbung der MOST Technologie, einem allgemeinen Multimedianetzwerkprotokoll und Anwendungsobjektmodell.

4. Das Konzept für die MOST Cooperation begann als eine informelle

kooperative Anstrengung im Jahr 1997. 1998 gründeten BMW, DaimlerChrysler, Harman/Becker und OASIS SiliconSystems die Cooperation formell, um die MOST Technologie zu standardisieren.

5. Im Zeitraum von 2000-2003 in der Entwicklung von Multimediakomponenten und Netzwerken für Fahrzeuge tätige Ingenieure hätten von der MOST Cooperation, ihrer Arbeit, gehört und auch gewusst, wie sie die MOST Cooperation kontaktieren können. Bis 2002 waren alle großen Fahrzeughersteller, Zulieferer und Systemarchitekten mit MOST involviert. Siehe **Anlage 7** bei 8, Henry Muyshondt, „MOST - Medienorientierte Transportentertainmentsysteme und Informationssysteminfrastrukturen in Automobilen“ (4. Dezember 2002). Die MOST Technologie war ein wichtiger Teil von Konferenzen der Branche, wie etwa dem Convergence 2000 Internationalen Kongress für Transportelektronik und wurde häufig in Referaten der Branche besprochen. Siehe **Anlage 8**, Akram Mufid, „Automobile Verbindungstechnologien für Multimedia-Subsysteme der Zukunft“, SAE Referat 2000-01-C028 (2000).

6. Ich habe das Medienorientierte Systemtransport (MOST) Spezifikationsrahmenwerk - Version 1.1 - 07, MOST Cooperation (1999) („MOST Rahmenwerk v1.1“) (beigefügt als **Anlage 1**), das Medienorientierte Systemtransport (MOST) Spezifikationsrahmenwerk - Version 2.1-00, MOST

Cooperation (Feb. 2001) („MOST Spezifikation v2.1“) (beigefügt als **Anlage 2**) und Medienorientierte Systemtransport (MOST) Spezifikation – Version 1.0-00 (Feb. 2001) („MOST Physical Layer-Spezifikation v1.0“) (beigefügt als Anlage 3) begutachtet. Diese Dokumente wurden vor Dezember 2002 direkt von der MOST Cooperation auf ihrer öffentlichen Webseite bereitgestellt.

7. Im Zeitraum von 2001-2003 hätte die MOST Cooperation jedem eine Kopie der MOST Spezifikation ausgehändigt, der darum gebeten hat. Siehe Anlage 7 bei 7. Um eine Kopie direkt von der MOST Cooperation zu erlangen, hätte jedes Mitglied der Öffentlichkeit die MOST Cooperation via E-Mail, auf dem Postwege oder per Telefon kontaktieren können und die MOST Cooperation hätte dann die Person auf den Download von ihrer Webseite verwiesen. Die MOST Cooperation verfügt über keine Aufzeichnungen, wie viele Kopien im Zeitraum von 2001-2003 angefragt oder auf ihrer Webseite bereitgestellt wurden. Wie ich im Folgenden jedoch erkläre, hätten die meisten Menschen diese Dokumente gratis von ihrer Webseite heruntergeladen, wo die MOST Cooperation diese in der Absicht zur Verfügung stellte, dass an der relevanten Dokumentation interessierte Personen in der Lage sein würden, die Dokumente zu finden und herunterzuladen. Zu diesem Zweck hielt die MOST Cooperation keine Suchmaschinen davon ab, den Inhalt ihrer Webseite zu katalogisieren, sondern stellte vielmehr sicher, dass der relevante Text auf jeder Seite erklärte,

was auf der Webseite zu finden war (z. B. nützlich für eine Suche nach Stichworten).

8. Entsprechend der üblichen Praxis der MOST Cooperation und auf in dieser Stellungnahme Bezug genommenen Dokumenten wurde das MOST Rahmenwerk v.1.1 sowie die MOST Physical Layer-Spezifikation v1.0 am 12. Oktober 2001 auf die öffentliche Webseite der MOST Cooperation und das MOST Rahmenwerk v2.1 am 16. November 2001 auf die Webseite der MOST Cooperation hochgeladen. Durch Klicken des entsprechenden Links konnte jedes Mitglied der Öffentlichkeit eine Kopie von der öffentlichen Webseite der MOST Cooperation herunterladen. Es ist die übliche Praxis der MOST Cooperation, Links zu den unterschiedlichen Teilen der MOST Spezifikation gratis bereitzustellen. Die MOST Cooperation speicherte das Datum, an dem ein Teil einer Spezifikation oder eine Aktualisierung zu diesem Teil auf die öffentliche Website hochgeladen wurde, neben dem Link, wie nachfolgend in **Anlagen 4 und 5** dargestellt.

9. Die im Zeitraum von 2001-2002 im Internetarchiv archivierten Kopien der Webseite der MOST Cooperation. Archivierte Kopien verschiedener Webseiten zeigen die Datumsangaben, zu denen MOST auf seine öffentliche Webseite verschiedene Dokumente hochgeladen hat, einschließlich des MOST Rahmenwerks v1.1 (Anlage 1), der MOST Spezifikation v2.1 (**Anlage 2**) sowie

die MOST Physical Layer-Spezifikation v.1.0 (Anlage 3). Ich habe das Internetarchiv in der Vergangenheit verwendet, und es ist mir bekannt, dass das Internetarchiv das Datum, an dem eine bestimmte Webseite archiviert wurde, in der URL der archivierten Kopie beinhaltet.

10. **Anlage 4** ist die Dokumente & Downloads Seite des MOST Rahmenwerks von der Website der MOST Cooperation. Das Internetarchiv zeichnete diese Kopie, wie aus dessen URL hervorgeht, am 30. November 2002 auf.[1] Anlage 4 zeigt, dass das MOST Rahmenwerk v.1.1 (**Anlage 1**) am 12. Oktober 2001 auf die öffentliche Webseite der MOST Cooperation hochgeladen wurde.

11. **Anlage 5** enthält die Spezifikationsdokumente und Downloadseite der MOST Cooperation, aufgezeichnet am 18. Dezember 2001, wie aus dessen URL hervorgeht.[2] Anlage 5 zeigt, dass die MOST Spezifikation v2.1 (**Anlage 2**) am 16. November 2001 öffentlich hochgeladen wurde und die MOST Physical Layer-Spezifikation v1.0 (Anlage 3) am 12. Oktober 12 2001 hochgeladen wurde.

12. Als eine Person, die in der Branche zu dieser Zeit tätig war, sind die

---

[1] Die URL der archivierten Kopie ist https://web.archive.org/web/20021130173527/http://www.mostcooperation.com:80/downloads/Specifications/MOST%20Framework/

[2] Die URL der archivierten Kopie ist https://web.archive.org/web/20011218195318/http://mostcooperation.com:80/downloads/Specifications/

Hochladedaten der Anlagen 4 und 5 konsistent mit meiner Erinnerung der Arbeit der MOST Cooperation zu dieser Zeit. Im Oktober 2002 begann ich meine Tätigkeit am Forschungszentrum Informatik in Karlsruhe (FZI), wo ich für Kooperationen mit der Wirtschaft und für Technologietransferprojekte zuständig war. Diese Arbeit erforderte es von mir, die MOST Spezifikation durchzusehen, und ich erinnere mich daran, die MOST Spezifikation zu dieser Zeit von der MOST Cooperation Webseite heruntergeladen zu haben.

13. Obwohl die MOST Cooperation nicht über eigene archivierte Kopien ihrer Webseite von 2001 und 2002 verfügt, was über 15 Jahre zurückliegt, habe ich keinen Grund anzunehmen, dass es sich bei den Anlagen 4 und 5 um irgendetwas anderes als akkurate Aufzeichnungen der Webseite der MOST Cooperation zu dieser Zeit handelt. Des Weiteren glaube ich, dass eine Person, die die Webseite der MOST Cooperation besucht, nachdem Anlagen 1-3 hochgeladen wurden (also am 12. Oktober 2001 für Anlagen 1 und 3 bzw. 16. November 2001 für Anlage 2), die Links zu diesen Dokumenten gesehen haben würde und in der Lage dazu gewesen wäre, diese herunterzuladen, wie ich es zu jener Zeit tat. Ich habe keinen Grund von anderem auszugehen.

14. Ein Grund, dass die MOST Cooperation alle Teile der MOST Spezifikation, einschließlich des MOST Rahmenwerks v1.1, MOST Spezifikation v2.1 sowie die MOST Physical Layer-Spezifikation v1.0 (Anlagen

1-3) der Öffentlichkeit allgemein verfügbar machte, war, dass die Kollaboration für Automobile Multimedia-Schnittstellen (AMI-C) es erforderlich machte. Bei AMI-C handelt es sich um ein Konsortium der weltgrößten Autobauer. Im November 2000 verkündete AMI-C, dass es die Verwendung von MOST als einem allgemeinen Hochgeschwindigkeitsbus für Autobauer und Lieferanten befürworten würde, tat dies jedoch erst, nachdem die MOST Cooperation ihre Spezifikation offen und der Öffentlichkeit verfügbar gemacht hatte.

15. Für mehr als ein Jahr vor der öffentlichen Unterstützung durch die AMI-C hatten Autobauer MOST als einen Kandidaten für die Befürwortung gesehen. Das AMI-C fand jedoch, dass MOST die Anforderungen der Zielvereinbarung des AMI-C nicht erfüllt, welche verlangt, dass Technologien „offen und lizenzfrei“ seien sollten. Infolgedessen verkündetete MOST im Oktober 2000, dass es seine Gebühren streichen und alle Teile der MOST Spezifikation offen und der Öffentlichkeit zur Verfügung stellen würde. Siehe **Anlage 6**, Charles J. Murray, Autobauer wählen MOST als Hochgeschwindigkeitsbus im Auto aus,“ EE Times (13. November 2000).

16. Durch Unterzeichnen dieser Stellungnahme erkenne ich an, dass die Erklärung als Beweiszwecken dienendes Zeugnis in einem Verfahren vor dem Ausschuss für Patentverfahren und Widerspruchsverfahren des Amts für Patente und Warenzeichen der Vereinigten Staaten eingereicht wird. Ich erkenne ebenso

an, dass ich Gegenstand einer Vernehmung durch die Gegenseite, beschränkt auf die in dieser Erklärung besprochenen Themen, sein kann. Bei Empfang zumutbarer und angemessener Bekanntgabe, dass solch eine Vernehmung durch die Gegenseite von mir erfordert wird, werde ich mich bereit erklären, zu erscheinen, und werde zwecks solch einer Vernehmung durch die Gegenseite zu zumutbarer Zeit an zumutbarem Ort innerhalb der Vereinigten Staaten erscheinen.

Alle hier aufgrund eigenen Wissens gemachten Stellungnahmen sind wahrheitsgemäß und alle auf Informationen und Überzeugungen beruhenden Stellungnahmen werden für wahr gehalten. Des Weiteren bin ich mir im Klaren darüber, dass diese Stellungnahmen in dem Wissen gemacht werden, dass wissentliche Falschangaben und dergleichen gemäß 18 U.S.C. § 1001 mit Geld- oder Freiheitsstrafe, oder beidem, bestraft werden können. Ich erkläre unter Androhung von Strafe bei Unrichtigkeit die Richtigkeit des Vorhergehenden.

Ausgeführt am 11. Mai, 2018 in Ötigheim, Germany.

Wolfgang Bott

Dr. Wolfgang Bott

BEFORE THE UNITED STATES PATENTS OFFICE

In re Inter Partes Review of:	)
U.S. Patent No. 7,489,786	)
Issued: February 10, 2009	)
Application No.: 10/316,961	)
E Filing Date: December 11, 2002	)

For: Audio equipment integration system FILED VIA E2E

STATEMENT BY WOLFGANG BOTT

IN ACCORDANCE WITH 37 C.F.R (CODE OF FEDERAL REGULATIONS). § 1.68

I, Dr. Wolfgang Bott, hereby declare the following:

1. I am currently the Technical Coordinator at the MOST Cooperation headquartered in Karlsruhe, Germany. My professional responsibilities include the coordination of working groups and the ongoing development of the specification. A part of my job involves promoting the MOST specification, encouraging others in the sector to apply the MOST specification and leading others in the sector to where it can be found.

2. 2003 I began working for MOST Cooperation as an advisor, and in 2005 I became the Technical Coordinator of MOST Cooperation. Due to my length of service at the MOST Cooperation, in general I am familiar with the history, its business

operations and practices, both before and after I joined it. I make this attestation on the basis of my own personal knowledge.

3. The MOST Cooperation is made up of a partnership network of car manufacturers, equipment manufacturers, systems architects and suppliers of key components. The MOST Cooperation is responsible for the ongoing development and promotion of the MOST technology, a general multimedia network protocol and application object model.

4. The concept for the MOST Cooperation began as an informal cooperative effort in 1997. In 1998, BMW, DaimlerChrysler, Harman/Becker and OASIS SiliconSystems formally established the Cooperation in order to standardise MOST technology.

5. In the period from 2000-2003, engineers working on the development of multimedia components and networks for vehicles had heard of the MOST Cooperation, and its work and knew how to contact the MOST Cooperation. By 2002, all the major vehicle manufacturers, suppliers and systems architects were involved with MOST. Refer to **annex 7** with 8, Henry Muyshondt, "MOST - Media-oriented transport entertainment systems and information system infrastructures in cars" (4 December 2002). The MOST technology was an important part of conferences in the sector, for example the Convergence 2000 international conference for transport electronics, and it was frequently discussed in reports within the sector. Refer to **Annex 8,** Akram Mufid, "Car connection technology for multimedia sub-systems in the future", SAE Conference 2000-01-C028 (2000).

6. I have provided the media-oriented system transport (MOST) specification framework - Version 1.1 - 07, MOST Cooperation (1999) ("MOST Framework vl.l“) (included as **annex 1),** the media-oriented system transport (MOST) specification framework - Version 2.1-00, MOST Cooperation (Feb. 2001) ("MOST specification v2.1“) (included as **annex 2)** and media-oriented system transport (MOST) specification - Version 1.0-00 (Feb. 2001) ("MOST Physical Layer specification vl.O“) (included as annex 3). These documents were made directly available before 2002 by the MOST Cooperation on its public website.

7. In the period from 2001-2003, the MOST Cooperation handed out a copy of the MOST specification to anyone who asked for one. Refer to annex 7 with 7. In order to obtain a copy directly from the MOST Cooperation, every member of the public was able to contact the MOST Cooperation publicly by e-Mail, by post or telephone, then the MOST Cooperation would refer the person to download from their website. However the MOST Cooperation does not have any records of how many copies were requested in the period from 2001-2003 or made available via its website. As I will go on to explain, the majority of people downloaded this document free of charge from the website where the MOST Cooperation had made it available, so that people who were interested in the relevant documents would be able to find the documents and download them. To this end, the MOST Cooperation did not prevent any search engines from cataloguing the contents of its website, rather it made sure that the relevant text on each side explained what could be found on the website (e.g. beneficial for a search by key

words).

8. In line with the established practice of the MOST Cooperation and the documents in relation to this statement, the MOST framework V. 1.1.was uploaded as well as the MOST Physical Layer specification vl.O on 12 October 2001 on the public website of the MOST Cooperation and the MOST framework v2.1 on 16 November 2001 on the MOST Cooperation's website. By clicking on the relevant links, every member of the public could download a copy of the public website of the MOST Cooperation. It is standard practice for the MOST Cooperation to provide links to the various parts of the MOST specification free of charge. The MOST Cooperation records the date on which part of a specification or an update of this part was uploaded onto the public website next to the link, as is shown in the following **annexes 4 and 5** .

9. The copies of the website of the MOST Cooperation archived in the internet archives in the period from 2001 -2002. Archived copies of different websites show the information about the dates that MOST uploaded the different documents onto its public website, including the MOST framework vl.l (annex 1), the MOST specification v2.1 **(annex 2)** and the MOST Physical Layer specification v.1.0 (annex 3). I have used the internet archive in the past and I am aware that the internet archive contains the date on which a specific website was archived within the URL of the archived copy.

10. **Annex 4** is the documents & downloads website of the MOST framework of the MOST Cooperation. The intenet archive records this copy which shows this URL

on **30** November **2002** . **1** Annex **4** shows that the MOST framework v. **1.1 (annex 1) was uploaded** on **12** October **2001** on the MOST Cooperation public website.

11. **Annex 5** contains the specification documents and download website of the MOST Cooperation, logged on 18 December 2001, as shown on the URL.2 annex 5 that the MOST specification v2.1 **(annex 2) was uploaded and made public** on 16 November 2001 and the MOST Physical Layer specification vl.O (annex 3) was uploaded on 12 October 2001.

12. As a person who was working in the sector at that time, the

---

[1] The URL of the archived copy is https://web.archive.org/web/20021130173 527/http ://www.mostcooperation. com: 80/ down1oads/Specifications/M O S T%20 F rain e work/

[2] The URL of the archived copy is https://web.archive.Org/web/20011218195318/http://mostcooperation.com:80/downloads/Specifications/

---

upload dates of annexes 4 and 5 are consistent with my memory of the work of the MOST Cooperation at that time. In October 2002 I began working at the Forschungszentrum Informatik (FZI) (Information Technology Research Centre) in Karlsruhe, where I was responsible for cooperation with the economy and for technology transfer projects.. My work required me to review the MOST specification, and I remember that at that time I downloaded the MOST specification from the MOST Cooperation website.

13. Although the MOST Cooperation does not have its own archived copies of

its website from 2001 and 2002, here we are going back more than 15 years, and I have no reason to believe that what is shown in annexes 4 and 5 are anything other than accurate loggings of the MOST Cooperation at the time. Moreover, I believe that a person visiting the MOST Cooperation website, then downloading annexes 1-3 (on 12 October 2001 for annexes 1 and 3 or 16 November 2001 for annex 2) would have seen the links to these documents and would have been capable of downloading them, as I did myself at the time. I have no reason to think otherwise.

14. One reason for the MOST Cooperation making all of the parts of the MOST specification widely available to the public, including the MOST framework vl.l, MOST specification v2.1 and the MOST Physical Layer specification vl.O (annexes 1-3) was that the collaboration for automotive multimedia interface (AMI-C) required it. AMI-C involved a consortium of the largest car manufacturers in the world. In November 2000, AMI-C announced that it would be endorsing the use of MOST as a comprehensive high-speed bus for car manufacturers and suppliers, although they only did so after the MOST Cooperation had made their specification public and available to the general public.

15. For over a year before the public support from the AMI-C, car manufacturers had seen MOST as a candidate to be endorsed. However AMI-C found that MOST did not fulfil the requirements of the agreement on objectives of AMI-C; these required that technologies should be "open and in the public domain". As a result of this, MOST announced in October 2000 that it would stop charging and would make

all parts of the MOST specification open and available to the general public. Refer **to annex 6,** Charles J. Murray, "Car manufacturers choose MOST as the high-speed bus in cars" EE Times (13. November 2000).

16. In signing this statement, I recognise that the declaration will be submitted as testimony and used in evidence in a procedure before the board for patent procedure and appeals of the United States Patents Office. I also recognise that I may be subject to interrogation by the counterparty, limited to the subjects discussed in this declaration. If I receive fair and reasonable notification that I am required for questioning by the counterparty, I declare that I am prepared to appear, for the purposes of such questioning by the counterparty, at a reasonable time and in a reasonable place within the United States. All of the attestations made here by virtue of my knowledge are truthful and all attestations and all statements based on information and beliefs are true. Furthermore I am aware that this statement is made in the knowledge that knowingly providing false information and the like is punishable under 18 U.S.C. § 1001 by a financial penalty or prison, or both. I declare the correctness of the above, under penalty of punishment for falsification.

Ausgeführt am 11. Mai, 2018 in Öhringen, Germany.

Wolfgang Bott

Dr. Wolfgang Bott

Annex 1

MOST

Media **O**riented **S**ystems **T**ransport

Multimedia and Control Networking Technology

MOST Specification Framework Rev 1.1

Version 1.1-07

Intellectual Property

Trademarks

All trademarks used in this document are proprietary of their respective owners. MOST and OptoLyzer are internationally registered trademarks of Oasis SiliconSystems AG.

Patents

There are a number of patents and patents pending on the MOST technology. The rights to these patents are not granted without any specific agreement between the users and the MOST Cooperation.

Support and further Information

For more information on the MOST Technology, please contact:

MOST Cooperation
Administration
P. O. Box 4327
D-76028 Karlsruhe
Germany

Tel: (+49) (0) 721 966 50 00
Fax:(+49) (0) 721 966 50 01

E-mail: contact@mostcooperation.com
Web: www.mostcooperation.com

Table Of Contents

Bibliography

Number	Document
[1]	MOST Specification Framework
[2]	MOST Specification
[3]	MOSTHighProtocol Specification
[4]	MOSTNetServices „Basic Layer“; User Manual and Specification
[5]	MOSTNetServices „Application Socket“; User Manual and Specification
[6]	FOT Datasheet
[7]	MOSTTransceiver Datasheet
[8]	MOSTFunctionCatalog

1 Introduction

1.1 Motivation

The motivation for the MOST (Media Oriented Systems Transport) technology comes from the requirement for a low cost high speed multimedia peer to peer network, that would not need a PC or any other central intelligence to operate in order to be cost effective, flexible, reliable and future proof. The option of having optical media as the physical layer has always been one of the main considerations for design of the MOST network. Our definition of multimedia includes everything from audio, video, telecommunications to data processing and control. One of the essential needs in such an environment is the support for true plug and play functionality (into and out of the network).

MOST network technology is a versatile, high performance, and low cost multimedia network technology based on synchronous data communication. Ideal for real-time applications such as CD quality audio, surround sound, and high quality video, it also supports control data and burst-type data transfers for both latency sensitive and latency insensitive network devices. It can operate with or without a PC, in a single or multiple master environment and with as many as 64 nodes providing all elements of plug and play.

1.2 Objective of the Specification

The objective of the entire specification is to describe the MOST system in terms of physical layer, transport layer, link layer, network management and the programming interface required to develop and build systems and devices that are compliant with this standard. The goal is to provide all information needed to make inter-operable devices in an open architecture but still leaving enough room for product and market differentiation without losing compatibility.

1.3 Scope of this Specification Framework

This specification framework is mainly targeted to all those readers, who want to get an introductory overview of the MOST System and its abilities. It provides valuable information to device developers, OEM's and System integrators, but also to independent hardware and software platform architects.

This overview can be used for system evaluation, product and system planning. For more information please refer to detailed specification documents (see below).

1.4 MOST Cooperation

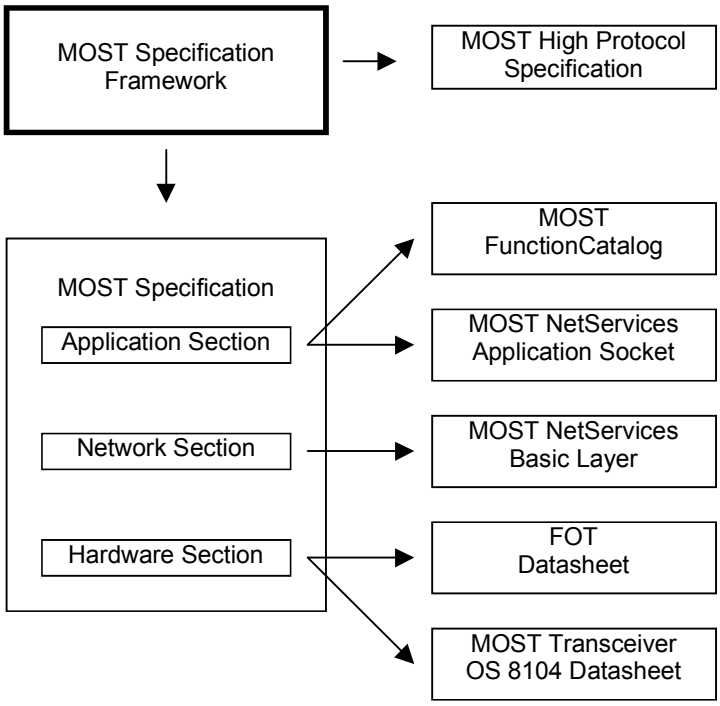
The MOST Cooperation is based on a partnership of Car Makers, Set Makers, and System Architects. Goal of the MOST Cooperation is, to define and develop a common multimedia network protocol and object model. All members of the MOST Cooperation get access to the technical information and to the specifications that are the result of the common work. Since verification of the various results is also a part of MOST Cooperation's work, their members have access to a reliable basis for their development activity. For generating products based on the information of the common information pool, licenses are required. Licenses can be obtained by every member of the MOST Cooperation. Licensing and handling of the intellectual property is task of the MOST Cooperation. For getting more information about MOST Cooperation in general, and about the joining of MOST Cooperation, please refer to the contact address to be found on page 2.

1.5 How to read this document

This document provides an overview of the MOST technology for all readers. More detailed information defining the MOST standard can be found in the associated documents:

- MOSTHighProtocol Specification
- MOST Specification
- MOSTNetServices „Application Socket“
- MOSTNetServices „Basic Layer“
- FOT Datasheet
- MOSTTransceiver Datasheet
- MOSTFunctionCatalog

This is the structural overview of the MOST Specification documentation:



2 Background

2.1 Evolution of the MOST Technology

MOST technology was originally developed for multimedia applications in the automotive environment, where an extremely robust, yet cost-effective network is needed. MOST networks, are replacing the bulky and expensive wiring harnesses that automobile manufacturers have used in the past to satisfy their multimedia connectivity requirements. The Plastic Optical Fiber based MOST Network not only provides substantially higher performance than the old wiring harnesses, but is more robust (no ground loops, etc.) at lower cost! Increases in the speed of the network and other improvements have now made the system appropriate for other environments in addition to automotive, such as multimedia PC systems and home networks. The continuing acceptance of MOST solutions will ensure that a robust, high performance technology will be available at lower costs to the consumer electronics market.

2.2 Main Goals of the MOST Technology

The MOST technology is specified to be an industry de facto standard for low cost, high bandwidth data communications in consumer, telecommunications and computing applications based on plastic fiber optics as a transportation layer. The following goals were essential in the definition of the architecture of the MOST System:

Applicable to consumer applications with or without the existence of a central control or PC
Low-cost solution with data rates up to 24.8Mbps
Optimized for use with Plastic Optical Fiber (POF)
Ease of use with excellent reliability
Full support for real-time audio and compressed video
Asynchronous and synchronous data transfer
Open protocol interface for use with a variety of protocols
Virtual network management embedded in the system
Integration into commodity technology
Widespread availability of key components
Scalability at the device level
Provide device specification for easy implementation
Network functions are as transparent as possible

2.3 Speed Requirements

In consumer applications the biggest concern in terms of success of a product or system is cost. However, the compromise between cost and performance has to be considered seriously. The following table shows the speed requirements of different applications and how MOST fits into this picture.

Cluster	Bandwidth	Physical Interface	Applications	Attributes
Control	10-100kbps	Man Machine Interface Interactive Devices Automatic Control System Control	System and Device Control and status report	Asynchronous Data Fair Arbitration Low Latency
Data Transfer	1-10Mbps	Hard Disks CD-ROM Memory Cards Data Service Area Network	Picture and other Data base transfer as well as Data communication such as RDS, Internet etc.	Burst Data Packet Oriented Asynchronous
Real-time	10 - 500kbps 0.5-4Mbps 1-12Mbps 2-50Mbps	Compressed Audio Audio Compressed Video Uncompressed Video	Radio Information, Digital Radio Infotainment, Communication, Navigation, Video CD TV, Cameras and CD Video	Continuos Bit stream Synchronous Transfer Constant or variable
	5-400Mbps	high resolution Video	digital TV and digital Video	Variable Bit rate

2.4 Features

MOST technology employs a synchronous approach in order to provide a low overhead, low cost network interface to even the most simple multimedia devices. It supports such devices as analog/digital converters for microphones and digital/analog converters for speakers with low intelligence and no buffering capability. Removing the need for buffering is a key feature of the MOST technology. All of this capability is provided without any compromise in signal quality.

MOST technology provides more complex DSP-based devices with all of the control and multimedia information necessary to fully utilize their capabilities which in turn maximizes the flexibility of the overall system. Equipment such as multimedia computers, analog audio gateways, multimedia CD players, hi-fi audio equipment, telecommunications terminals, video players, TV sets, satellite receivers, set top boxes, etc. can all be networked to interact at the lowest possible cost.

Since most equipment interconnected with this kind of network is capable of digital signal processing, the network is structured in a way that these applications can be implemented very efficiently. Multimedia related features, such as bandwidth allocation and peer-to-peer communication make it ideal for a serial multimedia consumer network, without the need of a central host. The current version of MOST technology can support up to 15 uncompressed, stereo CD-quality audio channels, up to 15 MPEG1 audio-video channels, or several MPEG2 video + audio channels (depending upon the MPEG2 implementation). In addition to the multi-channel source data, additional bandwidth for control, communication and asynchronous applications is always present.

The key features of the MOST technology are:

- **Ease of Use**
  simple connectors
  no hum loops, no radiation
  plug and play; self identifying devices with auto initialization
  dynamically attachable and re-configurable devices
  virtual network management including channel allocation, system monitoring addressing and power management

- **Wide Application Range**
  applications from a few kbps up to 24.8Mbps
  high degree of data integrity with low jitter
  support of asynchronous and synchronous data transfer
  support of multiple masters
  supports up to 64 devices
  simultaneous transmission of multiple data streams such as control, packet and real-time information
  devices can be constructed out of multiple functions
  low overhead due to embedded network management

- **Synchronous Bandwidth**
  Synchronous channels provide guaranteed bandwidth with no buffering required
  up to 24Mbps synchronous data throughput

- **Asynchronous Bandwidth**
  variable asynchronous data throughput
  up to 14.4Mbps asynchronous data
  dedicated control channel with more than 700kbps

- **Flexibility**
  wide range of real-time channel sizes and packet sizes
  remote operation and flow control
  variable arbitration mechanisms
  protocol independent

- **Synergy with consumer and PC industry**
  operates with or without PC
  consistent with PC streaming and the Plug and Play standards

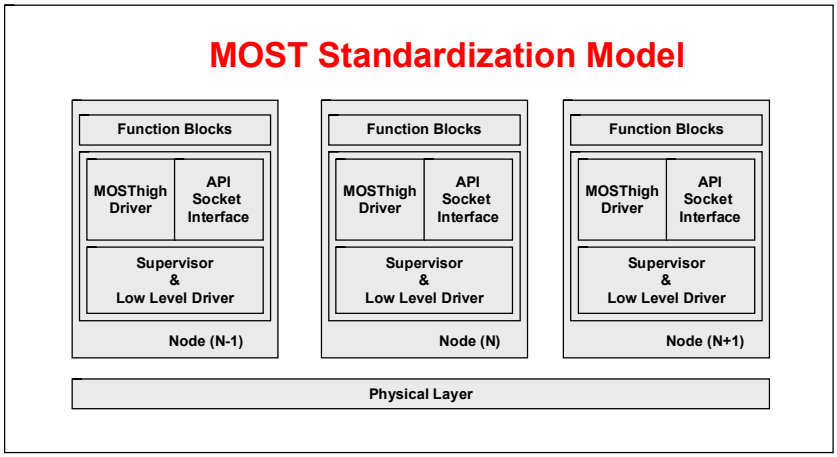
- **Low implementation cost**
  Low cost sub-channel at 700kbps
  optimized for implementation in consumer devices
  suitable for implementation in low cost peripherals
  Low cost cable and connectors
  Low cost integrated circuits

2.5 Compatibility

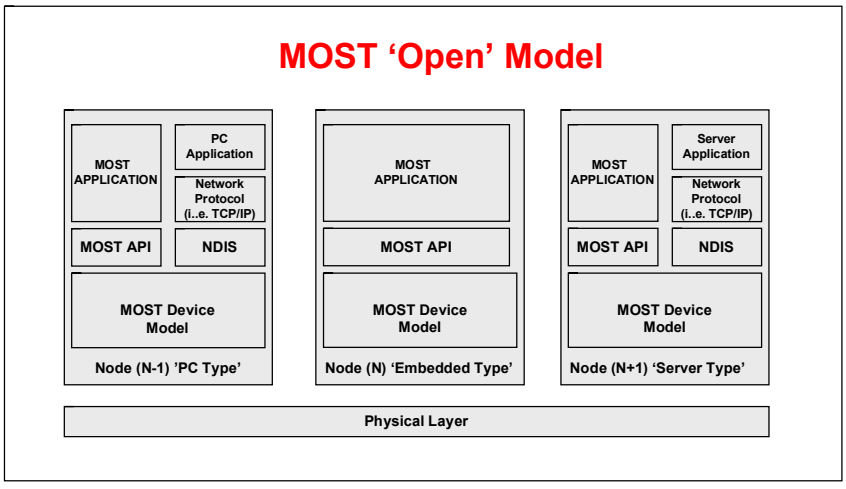
A family of network transceiver IC's will provide easy access to analog and digital audio I/O as well as to standard data formats such as I^2S, S/PDIF, AES/EBU, I^2C, SPI and others to operate with A/D - D/A converters, CD decoders and digital signal processors.

A MOST network can be used in conjunction with a number of different protocols. Since all basic methods required for source and control data communications are accommodated, the system is very flexible in terms of compatibility with a number of protocol layers.

A MOST network is capable to combine different nodes, independent of the internal structure of that nodes. So on one hand it is possible to combine nodes of the same kind, e.g. such nodes based on an embedded approach:



On the other hand, a mixing of nodes based on different approaches and with different internal structures is possible too:



3 Architectural Overview

This section describes the MOST System and its architecture as well as its key concepts. MOST is a peer-to-peer point-to-point network that can be implemented in a ring, star or daisy chain topology. The attached devices share different data channels for synchronous and asynchronous data transmission. Different access mechanisms are provided for different data types (e.g. stream, asynchronous (packet), control).

Since the MOST System is very flexible, it offers the possibility to handle a certain administrative task in different ways. There are two main approaches:

- Centralized approach
- Decentralized approach

The Centralized approach concentrates the handling of a special administrative task on one node in the network. Any other node that needs this service, has to contact this central node.

The decentralized approach does not need any central instance.

3.1 MOST System Description

A MOST System is described by the following definition areas:

MOST Interconnect
MOST System Services
MOST Devices

MOST Interconnect is the way that communications during system start up are established. During this procedure all devices get their unique device address. Communication between devices is possible after a successful interconnect procedure. Each device has an address relative to its physical position with respect to the timing master.

A synchronous bit stream interconnects the devices and provides synchronization to each node. A biphase coding scheme with frame and block synchronization is applied to re-synchronize and decode data at each device node.

The MOST System Services are defined by the three layers:

- Low Level System Services
- Basic Layer System Services
- Application Socket

Flexible real-time data routing through the network is managed automatically by the embedded low level system services. This algorithm handles the channel allocation and supplies the labels (channel ID's) for the requested channels as a result of an allocation request. Unused channels will be flagged and can be de-allocated in case of unplugging or de-installation of a device. Real-time data can have one or multiple destinations. All real-time (stream data) channels can be made available to all nodes within the entire system.

For control purposes each node has a network address and a node position address for on chip network system level management, such as fault reporting or delay detection. Control messages can be single-, group- or broadcast. The control channel has a transfer rate of more than 700kbit/sec.

Remote access to control functions and to the on chip registers of the network interface devices is available for standalone operation and diagnostics, so that built in functions can be remotely controlled and checked. Even external devices can be remotely controlled.

The MOSTNetServices, which comprise Basic Layer System Services and Application Socket, provide a standard interface with respect to the accessing of network management functions and the sending and receiving of the different kinds of data.

MOSTDevices can be anything from simple displays, up to complex applications. Since they are based on the MOST System Services MOSTDevices benefit from the comfortable standard interface. So devices have access to the messaging capabilities of a MOST system as well as to physical synchronous channels in groups of bytes and asynchronous packets. They can request a certain amount of bandwidth from the system for real-time data communication and can allocate packet and control data capacity at the same time.

3.2 MOST Devices

3.2.1 In General

MOST devices can inter-operate with each other in a peer-to-peer fashion. While source data streams are broadcast, control messages and packets can also be sent point-to-point. MOST devices can be anything from complex applications such as Man Machine Interfaces (MMI's), video players and receivers, simple displays, keypads and control i/o's. The embedded network management in combination with remote operation allows simple low cost devices to inter-operate within the system.

MOST Devices can be:

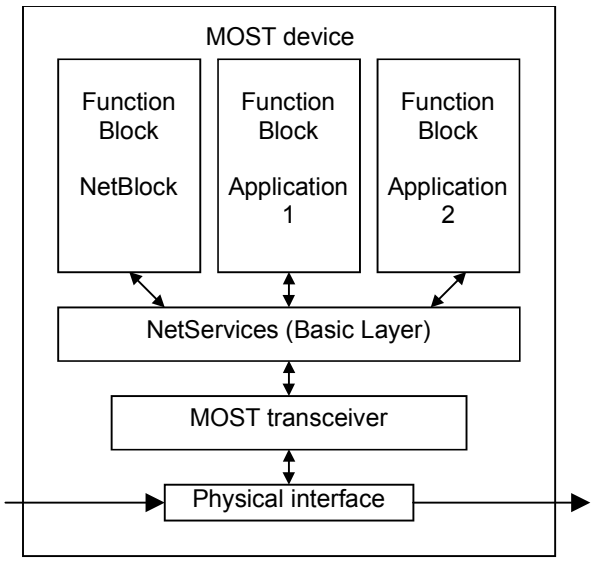
- Physical Devices such as physical i/o's
- Hubs which provide additional extension to the System. In a MOST network hubs can be transparent.

MOST Devices shall provide a standard interface in terms of their:

- Response to network management functions such as configuration, reset, channel allocation and fail safe operation
- Communications mechanism
- Physical Interface

3.2.2 Logical Approach

From the logical point of view, a MOST device consists of several function blocks, an interface level providing a standard interface (NetServices, Basic Layer) for communication between the function blocks and a unit providing MOST functionality (e.g. a MOST transceiver chip) and the physical interface to the MOST network.



A function block represents an application like e.g. a tuner, an amplifier, or a CD player. There is one special function block that handles functions related to the entire MOST device. This function block is called NetBlock, it is mandatory for all MOST devices. It is possible that a MOST device contains several different function blocks. The function blocks are based on the Application Socket of the MOST NetServices.

When implementing an application like e.g. a CD player, it needs functions like „Play“, „Stop“, „Eject“, or displaying time. So each function block contains a set of functions which are accessible from the external MOST world. There are two types of functions:

- Properties
- Methods

Properties are functions that provide determining or changing of the current status of the function block. Methods are functions that are started, and return a result after a certain time.

If a property is changed, this will trigger off an „Event“. The occurrence of an event will be notified to all devices which are registered in the event distributor.

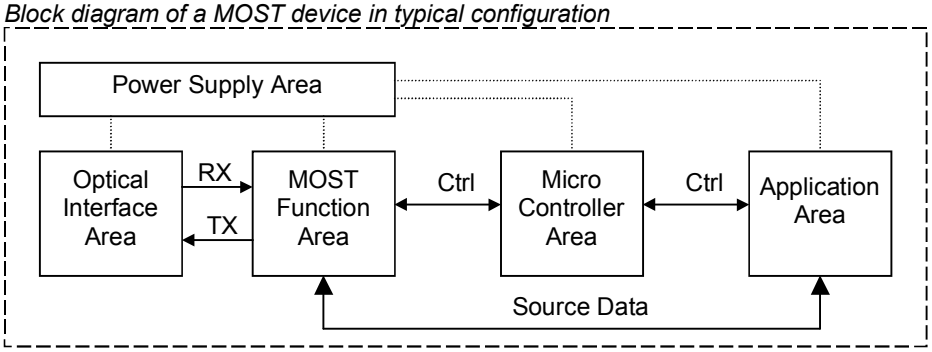
For being able to distinguish between the different function blocks (FBlocks) and functions (Fkt) of a device, each function and function block has a name, or an identifier (ID) respectively. When accessing functions, certain operations are applied to the respective property or method. The kind of operation is specified by the „OPType“. For distinguishing function blocks of the same type in a MOST network, there is another identifier called instance ID (InstID). So a protocol that gives access to a certain property in a function block looks like:

FBlockID.InstID.FktID.OPType(Data)

3.2.3 Hardware

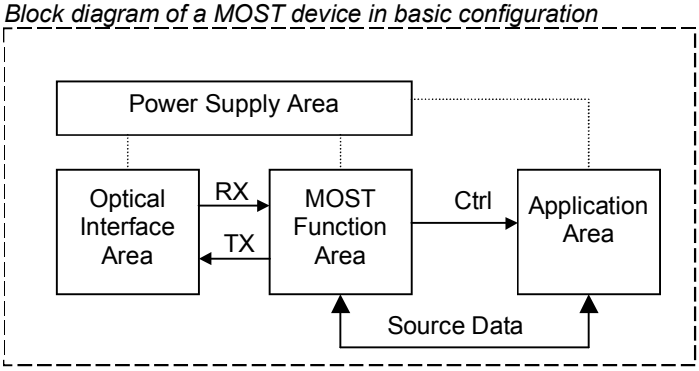
A MOST device can be implemented in two configurations. The standard application for a MOST device contains a micro controller.

Block diagram of a MOST device in typical configuration



For simple tasks, it is sufficient to combine e.g. a MOST transceiver and an optical interface in conjunction with a power supply. This operation mode is called „Standalone mode“. Then it is possible to remote control this device e.g. by a remote MMI. :

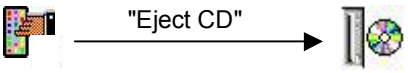
Block diagram of a MOST device in basic configuration



3.3 Data Types

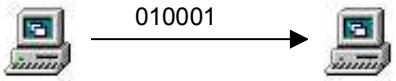
The MOST system supports a variety of data types such as control data, packet data and synchronous stream data. The basic types of data transfers support the following functions:

Control data transfer for device specific transfers and system management:

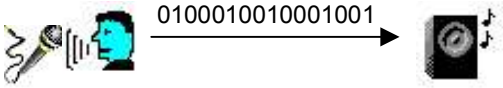


This kind of data is transported in parallel to the real - time and asynchronous data.

Bulk data / burst data transfer in asynchronous packets with variable bandwidth requirements:



Real-time data transfer which requires a guaranteed bandwidth to maintain data quality.



If real-time data is sent to the MOST network, all nodes have access to it. The number of nodes „listening“ to real-time data is only limited by the maximum number of nodes.

3.4 BUS Protocol

All bus transactions involve the transmission of a synchronous frame from a single frame generator or timing master. This can be any device within the system, however it needs to be determined which device will take over this function. The configuration of the frame generator can be done before system start up as for hard disk master slave devices. Typically this will be a Man Machine Interface device.

All slaves will lock to the bit stream generated by the frame generator by use of an internal PLL within each timing slave device. Timing slave devices must either be synchronized to the timing master or provide a means of synchronization by acting as a slave to the clock being regenerated from the bit stream. Typical transceivers will provide access to asynchronous and synchronous channels of the systems.

The network system management functions are supported by the bit stream and are embedded in the low level network management within each single node. This function is implemented with unique addressing during start up, channel allocation and system set up (e.g., synchronous versus asynchronous bandwidth allocation).

A remote network node operation function is supported on a bit stream block level. By this means standalone nodes can be remotely controlled without the need for any application implemented within those nodes (e.g., remote displays, keyboards, headphone jacks etc.).

The details of the Frame and Block structure of the bit stream are described in document [2].

3.5 Physical Interface

MOST technology complies with all requirements for different physical layers and is able to support complex topologies required for plastic optical fiber applications or infrared controlled systems. Since plastic fiber is the most promising medium for future home and multimedia networks, the MOST network is future proof and state of the art. Operation over the substantial node-to-node distances required for multimedia home networking is not an issue using plastic fiber and high speed LED's available today. The system is competitive in cost today compared to copper solutions proposed for similar applications, and has far more cost reduction potential over time. The MOST network can be configured in ring, double ring, star, tree or combined topologies and has a flexible plug-and-play structure with auto device detection. A detailed description of the proposed physical interface for MOST is provided in the MOST Physical Interface description.

The physical interface consists of the following signal lines:

RX - receive data line
TX - transmit data line

Optional lines could be:

Power -
Ground -
Wake_up - an optional wake up line for waking the MOST network

The wake up function in general is derived from sensing activity on the RX line. A receiver device should be able to detect activity and wake up the target device. As an additional option, an electrical wake up line might also be implemented. A MOST device would wake up on activation of that line. The power supply can come from anywhere since common mode problems are not expected due to the use of optical transmission techniques.

3.6 Power Management

Since power considerations will become more and more important, the system defines a zero power mode that can be implemented in all devices. In this mode, systems consume only a few microamperes.

The power management features of MOST make it suitable for power sensitive systems such as mobile applications (e.g. automotive, portable audio, video, communications).

3.7 MOSTTransceiver

The MOSTTransceiver OS8104 of OASIS SiliconSystems can handle all interfacing tasks between physical interface and the MOSTNetServices, and the Stream Services respectively. It contains the entire functionality of the MOST Low Level System Services, except physical interface (Media and connectors).

OS 8104 supports a variety of serial source data port formats, so many different hardware components, e.g. for audio I/O tasks, can be used in MOSTDevices. Interfacing to the transceiver is possible in serial or parallel.

A detailed description of OS 8104 is to be found in document [7].

3.8 Hardware Requirements

Each node is responsible for the following function:

- Clock recovery (or generation in case of master fail safe operation)
- Node address identification and decoding
- Management of control and data flow between devices
- Detecting system start up and reconfiguration
- Power management
- Initialization from non-volatile memory area
  Initialization from external EEPROMS or ROMS should be possible in network nodes to download system parameters and configuration setups.

3.9 Software Requirements

MOST system software manages communication and data flow between MOST devices. There are the following interactions between the MOST system and the device software:

- Physical addressing and network configuration
- Synchronous channel allocation and de-allocation
- Control data transfer
- Packet data transfer
- Power management
- System monitoring and management

3.10 System Integrity/Robust Operation

Several attributes of the MOST system contribute to its system integrity :

- Use of Plastic Optical Fiber provides electrical de-coupling and low bit error rate
- Parity over Frame detects bit errors together with PLL lock detection
- Ultra low jitter ensures high data quality (e.g. low THD + Noise in audio signals)
- Plug and Unplug event detection provides dynamic reconfiguration on the fly
- CRC protection over control frames, data packages and remote control commands
- Remote diagnosis provided using remote control commands
- Fair arbitration mechanism implemented in the control channel with different priority assignment
- Synchronous channel allocation with guaranteed bandwidth and data quality
- Asynchronous channel allocation using token arbitration

3.10.1 Error Detection and Handling

Error detection can be provided on different levels. Parity over Frame provides one of the essential levels of error detection. Together with the frame synchronization pattern, the parity detection and PLL error status flags transmission errors immediately. Control frames and packet data are protected by additional CRC and data transfer acknowledgment.

Error handling is provided by MOST System services. Software on application level can use this information for high level error management.

3.10.2 Fail Safe Mechanisms

There are several fail safe mechanisms defined for the MOST system. These are:

- Remote read and write function
- Watchdog at application (watchdog timer)
- electrical bypass
- network activity detection
- network status and line quality report
- emergency timing master function for error reporting

3.11 System Configuration

The MOST system allows devices to be attached to and detached from the MOST system at any time. As a result automatic configuration detection is a dynamic feature during MOST system operation.

3.11.1 Attachment of MOST devices

As soon as an additional device is attached to the MOST system a configuration change report message will be issued by the system. The Device Unique Addresses will change due to system reconfiguration and the number of devices in the network will change also. Any device can communicate with any other device after network configuration.

3.11.2 Detachment of MOST devices

When a MOST device has been removed from the system, a configuration change message will be issued by the system. The Device Unique Addresses will change due to system reconfiguration and the number of devices in the network will change also. Any device can communicate with any other device after network configuration.

3.12 MOST Topology

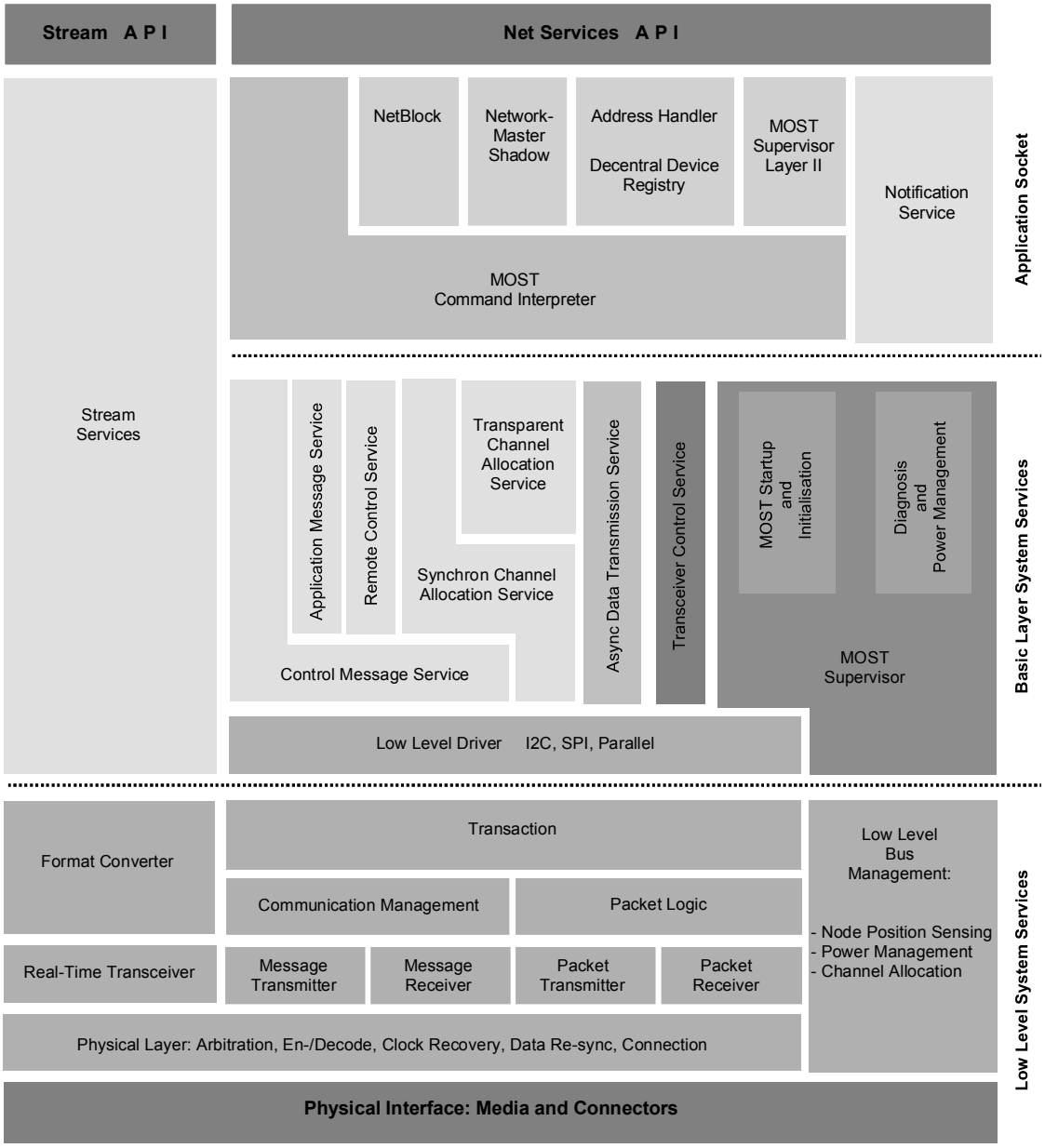
There are two basic system architectures supported by MOST Technology. The first is a one-way, point-to-point link requiring a simple transmitter on one side and a receiver on the other. This is an extremely cost effective, one-way synchronous digital connection for applications such as connecting a digital audio source to active speakers, or a digital video source to a monitor. The second basic system architecture is a network with a ring topology. This network can have as few as two nodes, effectively a full duplex point-to-point connection, or as many as 64 nodes. A number of variations of the ring topology are possible as well. For example, optical splitters can be incorporated into a POF Ring creating branches connected to receive-only nodes. Another variation is the “Star of Rings” configuration, used where centralized network management or access to all inter-node connections is an important consideration. Finally, in environments where system speed, reliability and minimization of downtime is paramount, a dual ring topology can be used.

4 MOST System Services

The MOST System Services provide all basic functionality to operate a MOST System. They are designed to offer a maximum of flexibility at the highest level of comfort and consist of :

- Application Socket
- Basic Layer System Services
- Low Level System Services
- Stream Services

Block Diagram of the MOST System Services



4.1 Application Socket

The Application Socket offers a wide variety of functions for building an application. Some of the functions are mandatory and some depend on the kind of application and are therefore optional. The Application Socket together with the Basic Layer System Services form a „basic device“, that contains all basic functionality a MOSTDevice needs. The Application Socket contains:

- MOST Command Interpreter
- NetBlock
- Network Master Shadow
- Address Handler/ De-Central Device Registry
- MOST Supervisor Layer II
- Notification Service

4.1.1 MOST Command Interpreter

The MOST Command Interpreter is based on the Application Message Service of the Basic Layer. It interprets and distributes received messages to the different function blocks of a MOST Device.

4.1.2 NetBlock

The Net Block is a function block which is mandatory for every MOST Device. It contains basic properties of the entire device (e.g. Addresses and available function blocks).

4.1.3 Network Master Shadow

The Network Master is a central instance (on highest level) in a MOST network, which handles administrative tasks like the „Central Registry“. Network Master Shadow is a local image of the Network Master which is needed for being able to receive messages of the Network Master. It is optional, since these functions can be handled in a de-central manner too.

4.1.4 Address Handler, De-Central Device Registry

If the actual logical MOST address of a function block is unknown, this optional service seeks it anywhere in the MOST network. It is possible to keep a local device registry, which then contains the respective logical addresses.

4.1.5 MOST Supervisor Layer II

MOST Supervisor Layer II provides initialization of a device on highest level, e.g. of the logical address.

4.1.6 Notification Service

This service organizes the sending of notification messages, which are sent if a property is changed.

4.2 Basic Layer System Services

The Basic Layer of the MOSTNetServices provides comfortable access on the Low Level System Services. The Basic Layer System Services consist of:

- MOST Supervisor
- Low Level Driver
- Control Message Service
- Synchronous Channel Allocation Service
- Transparent Channel Allocation Service
- Asynchronous Data Transmission Service
- Transceiver Control Service

4.2.1 MOST Supervisor

This service offers transceiver initialization and network startup on a higher level. Failure diagnosis and power management are also supported by MOST Supervisor. So Zero power mode can be initiated via the network or via and external control command. In this mode, the network is constantly checked for activity. As soon as there is activity, the device will power up. The error checking mechanism is able to detect line errors. As soon as a slave detects valid data it will report a valid line connection to the system.

A system first power-up event is used to signal the first power-on event (e.g., cold start of the system) to any application. As soon as the network is stable, a net-on event will be provided. As long as the PLL is not locked and the node is not initialized by any application the node will stay in bypass mode. A node initialization event will activate the node.

Two different services are provided for system monitoring. The network activity is constantly checked in each node. Any errors or malfunction of the network will be reported to the MOST Supervisor.

As a result the device will switch into timing master mode and send out its network position together with an error code. Similarly, a problematic timing master node will switch into slave mode.

In case the input of a node is lost without a previous shut down procedure, an emergency procedure will be executed.

4.2.2 Low Level Driver

The Low Level Driver provides an interface between microcontroller area and the MOST transceiver. It therefore offers different interface formats. For adapting to the respective micro controller (µC) environment, the Low Level Driver is complemented by user definable hardware specific functions.

4.2.3 Control Message Service

The Control message Service buffers normal control or system messages when sending and receiving. Error detection and notification is provided too.

4.2.3.1 Application Message Service

When sending, this service splits application messages (FBlockID.InstID.FktID.OPType(Data)) into several segments with the length of 17 bytes. When receiving, segmented messages are restored to their original structure. Messages are buffered. Error detection and notification are also part of the Application Message Service.

4.2.3.2 Remote Control Service

This service provides the sending and receiving of remote system messages, including error detection and notification.

4.2.4 Synchronous Channel Allocation Service

The Synchronous Channel Allocation Service is based upon embedded Channel Allocation which is part of the Low Level Bus management. It allocates or de-allocates resources for real time data streaming on the MOST network and „routes“ this data to the desired destination. It contains functions to make real time data handling comfortable, e.g. by providing a list of the offsets of all physical channel related to one logical channel with respect to the synchronous data stream.

4.2.5 Transparent Channel Allocation Service

This service provides allocating and de-allocating as well as routing of transparent data.

4.2.6 Asynchronous Data Transmission Service

Sending and receiving of asynchronous data is the task of this service. Errors are detected and notified.

4.2.7 Transceiver Control Service

This service provides access to configuration and address registers of the MOST transceiver.

4.3 Low Level System Services

The Low Level System Services consist of the sections:

- Physical Interface
- Physical Layer
- Low Level Bus Management
- Packet Logic
- Communication Management
- Transaction Level
- Real Time Transceiver
- Format Converter

4.3.1 Physical Interface

The physical interface provides mechanical connection between the Fiber Optic Receiver/ Transmitter (FOT) units and the Plastic Optic Fiber (POF). In addition to that, it connects the FOT units electrically with the MOST transceiver. The Physical Layer is optimized for the use of POF as the transfer medium, but copper cable (e.g. UTP, Coax) is permissible too.

4.3.2 Physical Layer

The physical layer as implemented in the MOST transceiver OS 8104, provides connection of the MOST devices. In addition to that, clock recovery, data de- and encoding as well as arbitration for asynchronous channels are implemented.

4.3.3 Low Level Bus Management

The Low Level Bus Management provides a set of functions, that handles power management as well as Channel Allocation for real - time data, and Node Position Sensing. In addition to that, it provides information for delay compensation. A message to a particular device, a group of devices, or all devices as a result of a status change of the INT pin is available in standalone mode too.

4.3.3.1 Addressing

As soon as the network has started up, all nodes will get a unique device identifier. The identifier can be used as a unique network address. All devices can be individually addressed after this procedure for functional verification, network operation and remote control. Installation or de-installation of devices may change the devices network address and at least the number of devices in the system. Therefore a reconfiguration procedure has to take place after unplug/ plug events. The unique device ID and the total number of devices on the network is available to the application.

4.3.3.2 Allocation Table

The Allocation Table is a structure that provides information about the physical channels and about their channel ID's grouping them into logical channels. The Allocate Logical Channel Request and De-allocate Logical Channel Request messages will change the contents of the allocation table.

4.3.3.3 Allocate Logical Channel Request

This function will allocate up to 8 physical channels and group them into one logical channel. A granted request results in a unique channel ID and the offset of the allocated physical channels inside the synchronous data stream. A denied request results in a report of the remaining number of available physical channels in the data stream.

4.3.3.4 De-allocate Logical Channel Request

A logical channel can be de-allocated using this function and its channel ID as a parameter. The corresponding physical channels will be available after a de-allocation event.

4.3.3.5 Initialize Allocation Service

This service will initialize the allocation table. This procedure is executed after system start up and after any changes of the boundary descriptor.

4.3.3.6 Allocation Table Distribution Service

The Allocation Table is distributed automatically in the background and is available to every node.

4.3.4 Packet Logic

Packet Logic controls sending and receiving of packet/ bulk data in Packet Transmitter and Packet receiver. It includes error detection for packet data.

4.3.5 Communication Management

Communication management controls sending, receiving and error detection of MOST control messages (normal messages as well as system messages).

4.3.6 Transaction Level

Provides access to data received either as packet data, or MOST control message, and to the transmit areas for sending data (asynchronous and control).

4.3.7 Real Time Transceiver

The Real Time Transceiver receives real time data from the external world, or sends this kind of data to the external world. In addition to that, it performs „Routing“ on real time data, that means it puts the data to the right destination (network or application).

4.3.8 Format Converter

The format converter converts a variety of serial source data formats (S/PDIF, Sony, Matsushita, I2S,...) into a format that can easily be transported via MOST network, or vice versa.

4.4 Stream Services

The Stream Services provide transport services for real-time Source data. This allows to handle Source data in the respective parts of the application area.

5 MOST High Protocol

MOST High Protocol uses some of the established mechanisms of Transmission Control Protocol (TCP) for providing secure data transmission via MOST Control Message Channel and Asynchronous (Packet) Data Channel. It is part of the MOSTNetServices and needs not to be re-implemented for each MOSTDevice that shall be developed.

TCP is often used for achieving reliable data transmission between two PCs. It guarantees safe data transmission with positive acknowledge and automatic repetition in case of failure.

An acknowledge is given after each block that was received. Acknowledgement after reception of a group of blocks is possible too. A timer, which is part of the protocol, initiates re-sending of a block after a certain time, if no acknowledge was given.

TCP is a duplex connection, that provides data exchange in both directions. Acknowledging of data within the same frame that contains the data, avoids wasting network bandwidth.

For sending bulk data and control data, the same header format is used, while the kind of data is indicated in the coding field of the header (maximum header size is 192bits). For avoiding data overflow in the receivers there is a „window field“ indicating the number of octets that can be received.

For MOST High Protocol, the header is reduced to the essential extent, and adapted to the MOST environment. MOST High Protocol is usable for providing secure data transmission within MOST Networks, it is not suitable for communication with the external world.

For more detailed information please refer to document [3].

6 MOST Frame Structure

The continuous bit stream data are biphase coded for high data integrity and precise clock recovery with low jitter as required for high quality conversion systems. This provides optimized data transmission via Plastic Optical Fiber and is a proven technology. The bit error rate is typically less than 10 e -10. The typical baud rate with a Sample Frequency of 44.1kHz is 45.1584Mbaud. One node provides the system clock (clock source) and all other nodes slave to this clock using their low jitter PLL (Phase Lock Loop).

6.1 Frame Generation

Frame generation is provided by a single device in the system also referred to as the timing master. This can be any device in the network but would typically be a control unit or man machine interface device.

6.2 Synchronization

All other devices in the network will slave their clocks to the timing master's frame and bit stream. A Phase Lock Loop (PLL) in each device is responsible for individual clock recovery within each node. A minimum amount of buffering is required in each node for phase synchronization in case of synchronous data transfer and for packet synchronization in case of asynchronous data transfer.

6.3 Communication Model

The communication model in a MOST system is based on peer-to-peer communication in combination with broadcasting abilities.

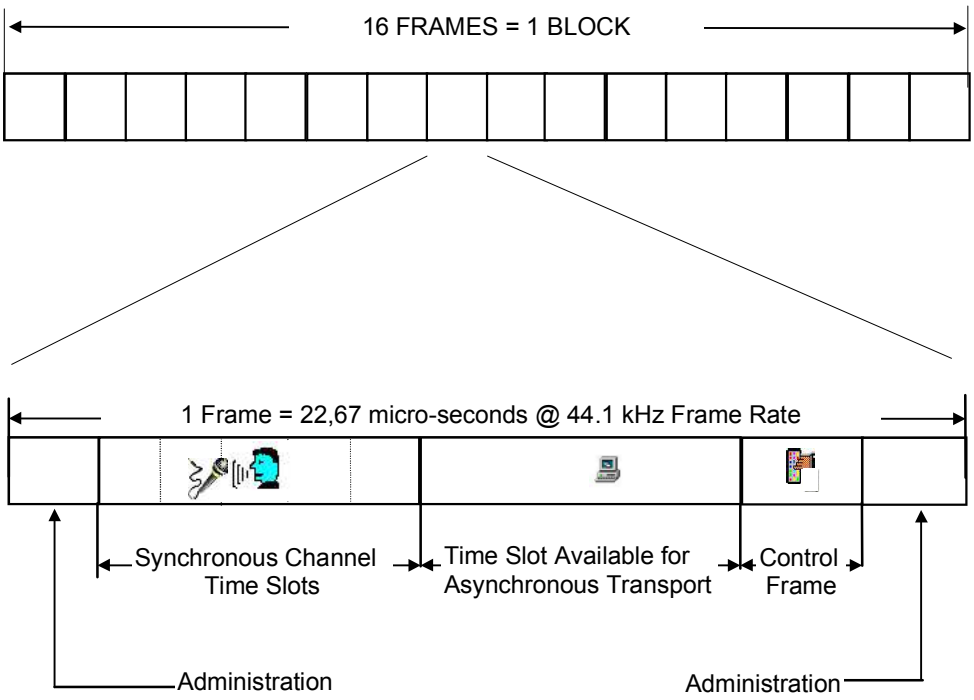
6.4 MOST Bit stream

The bit stream is optimized in such a way that processing is easy and maximum functionality is supported. This includes mechanisms for automatic channel routing, network delay detection and burst data channel management.

The MOST network technology defines an intelligent bit stream which is capable to provide all MOST network features as described above.

6.5 Block

Each block consists of 16 Frames running at the System Sample Rate as Frame Rate (typical 44.1kHz). The block boundaries are only relevant for Control Frame handling and Network Management.



MOST Block and Frame Structure

6.6 Frame Functionality

The MOST Frame structure is designed in a way that it provides maximum flexibility in terms of compatibility with a number of existing communication and data transport requirements without any drawbacks in implementation cost or processing overhead. It also allows easy re-synchronization, clock and data recovery at highest data quality and integrity. Built-in structures allow simple network management on the lowest layers avoiding overhead and cost shortcomings.

6.7 Frame Definition

The MOST frame consists of 5 sections. There are sections that contain administrative data for synchronization and data security. 60 bytes are reserved for transporting synchronous source data and packet data, 2 bytes are available for control messaging. For dividing up the 60 bytes between synchronous source data and asynchronous data, there is a boundary descriptor value which is transported in the administrative sections.

The smallest portion accessible for the user is one byte also referred to as a channel in case of synchronous data. This minimizes data overhead at high real time data throughput without loss of flexibility.

For a more detailed description of the MOST Frame, please refer to document [2].

6.8 MOST Data Channels

Sixty data bytes (15 quadlets) total are available for synchronous and asynchronous data. The number of synchronous and asynchronous quadlets is indicated by the Boundary Descriptor value described previously.

6.8.1 Synchronous Channel

The Synchronous Channel Time Slots are available for real-time data such as audio/ video or sensors and eliminate the need for additional buffering in analog-to-digital converters (and digital-to-analog converters) or in single speed CD devices for audio and video.

The access of these data is provided by Time Division Multiplexing (TDM) and allocation of quasi-static physical channels for a certain period of time (e.g., while playing an audio source). The bandwidth for such a channel can be adjusted by allocating any number of bytes to one logical channel. The maximum number of bytes available in a synchronous channel is 60 bytes/ Frame, which is corresponding to 60 x 8 bits or 15 stereo channels of CD quality Audio. The typical Frame rate is 44.100 Frames/ second.

The Routing Engine (RE) is used to route data to and from the appropriate sources or sinks within a node. Internal synchronization is provided to allow input data not to be phase aligned to the transceiver. The RE provides full flexibility in directing data from any source to any sink by just setting the appropriate value in the corresponding registers.

6.8.2 Transparent Channel

Some applications, however, may provide asynchronous data interfaces such as RS 232 and as a consequence require transparent asynchronous data transfer capability. In this case the asynchronous data lines can be oversampled at the synchronous channels and routed over the network to any other source data port.

6.8.3 Asynchronous Packet Transfer Data Channel

Another time slot is available for Asynchronous Data Transport as required for high frequency burst and packet data transport in applications such as hard discs or computer peer-to-peer communication. The access to this type of data is provided in a Token Ring manner.

Each node has fair access to this channel and its bandwidth can be controlled using the boundary descriptor in a step of four bytes (quadlets).
The maximum packet length on an asynchronous channel is 1014 bytes, the data on this channel are CRC protected.

Since the asynchronous data area is variable, it can take several frames to complete a message. The corresponding management such as arbitration and channel allocation is provided by the chip. The definition of the asynchronous message is described in more detail in [2].

A hardware CRC is provided. The CRC is calculated in the background and can be indicated in a register at the end of each asynchronous message.

6.8.4 Control Data Channel

The third portion of the bit stream is allocated to Control Frames as required for media control and other control data communication such as control of consumer devices and other control applications (i.e. home automation, process automation or car multimedia control). The protocol on this channel runs in a Carrier Sense Multiple Access (CSMA) manner offering predictable response times which are considered essential in an audio/ video control network. The bandwidth of this channel provides a data throughput of more than 3000 control and status messages per second. Sending and receiving messages is handled automatically by the chip. One complete message is buffered by the chip. Any important status changes will be flagged by interrupt. The data on the Control channel are protected by CRC and acknowledge mechanisms.

There are two kinds of Control messages. The „normal“ messages provide control of application, while the „system“ messages handle system related operations like resource handling, or remote access. During remote access, additional handshaking guarantees reliable remote operation. Each Control message transports 17 Bytes of user data.

Arbitration is provided automatically by the transceiver in case a node wants to send a message. In order to provide fair arbitration even at high bus loads a double arbitration mechanism ensures that an access will not be depending on communication load of upstream devices and the priority is not depending on the network position. Rejection of messages will be flagged and automatic retransmission can be used when switched on. The number of retries can be defined by the application software. An automatic CRC is also provided for this channel and can be read from a register.
In Standalone Mode, the MOST control messages can be used to indicate the occurrence of an interrupt.

7 Low Level System Services

Except the physical interface, the entire Low Level System Services are implemented in the MOST transceiver. This chip itself handles a great deal of network management tasks at the lowest level. The network management includes such functions as start up and shut down of the network, fail safe bypass, error reporting, power management and channel allocation.

Physical position sensing, network delay detection and node alive supervision are essential mechanisms provided within the transceiver core. Part of the initialization of the network is the node position sensing mechanism which provides a unique physical address for every chip in the network. This installation procedure is part of the network management and is done in the transceiver. A device can also be installed with a logical address within the network. The application will write the target logical address into the transceiver and the embedded network management verifies its unique existence. In case the address is already used by another device, the address will be rejected and the application will be notified. Once a device is installed and has a logical address it will retain its address until de-installation. All data requirements and channel connections are serviced during installation and dynamically present in the background during normal operation.

A device can be either active or passive. During active operation the request for data capacity comes from the application using a particular communication mode. As soon as a connection has been made the required data capacity will be provided by the network.
The application does not deal with any resource management within the network as long there is no resource conflict reported. Resource conflict will be detected as soon as the maximum network capacity of more than 24Mbits/ sec is not sufficient to serve the applications running on the system. In this case the lack of additional data capacity will be flagged.

The source data allocation algorithm is managed on the bit stream level. This approach results in fast response times (within the millisecond time frame) and high data throughput efficiency. Once a synchronous connection is built it will typically stay for the lifetime of the corresponding application. Burst data channels are managed on the frame level since their data capacity might change very quickly and the latency time to carry away such data is normally low.
The inherent network delay can be compensated using the network delay reporting mechanism making the information on relative network delay with respect to each channel available at every node.

The MOST architecture's high data capacity prevents the system from running into data capacity conflicts. This makes high level software administration and segmented transmission obsolete in nearly every application.

7.1 Automatic System Configuration and Start up

The majority of network management functions in a MOST network are handled automatically on a distributed basis and are embedded into the MOST transceiver itself. Since channel allocation, physical addressing, fault monitoring and zero power control is automatically provided by the system, the implementation of real world applications is very simple, system configuration is done automatically with a high level of network protection.

7.2 Hot Plug-in

Since MOST Technology is optimized for POF applications, hot plug-in is an easy task. No short circuit protection is required and physical plugging and unplugging can do no harm. On top of that, the use of basic on-chip mechanisms ensure the network's plug and play and hot plug-in capability. This includes channel allocation and physical position management.

A change in the configuration will be automatically detected and results in a new physical position sensing.

7.3 Synchronous Channel Allocation

Source data routing becomes an easy task, since all necessary functionality is supported by the bit stream structure and managed by the system. All information on currently used bytes and the position of free data channels is available on each node and handled virtually by the network and its MOST devices.
All source data channels are bytes. A logical channel may require a number of physical channels (bytes) depending on its real time data requirements. Logical channels can be clustered into groups at higher software layers. The maximum size of a logical channel that can be allocated at once is 64 bits.

The channel allocation is supported by a network level with an on-chip channel allocation algorithm. A channel allocation table is available within each node indicating the labels of source data channels (connection labels) available at a certain point in time. Since synchronous source data channels (bytes) are quasi static, two functions for allocation and de-allocation are available. Nodes that want to source data can identify the existence of free channels and request allocation from the network. Nodes that want to sink data can identify the correct channels by the connection label without the need to send messages to the source nodes. Up to 60 synchronous byte sized data channels are potentially available. The channel allocation can be changed during runtime. The entire channel building procedure has a maximum latency time of 25 ms.

7.4 Asynchronous Bandwidth Allocation

The boundary descriptor determining the bandwidth for asynchronous data can be set in a frame generator directly or via remote access. All timing slaves will automatically adapt to this frame format.

7.5 Physical Position Sensing

The node position relative to the Frame Generator is available in each node at any time, as soon as the network is running. The node position is used to determine the node's unique location (and address) in the network. The Frame Generator senses the number of nodes in the network and provides this information to the application. Using the node position for configuration sensing enables "hot" plug-in when dynamically reconfiguring the system.

7.6 Network Delay Detection

The delay in a network relative to the Frame Generator is not always directly related to the node position, since nodes can be active or passive. Active nodes are those that source data into the network, whereas sink-only devices are passive. The network delay detection provides the accurate number of frame delays to each node and can be used for delay compensation. As in the case of the network position, the Frame Generator always knows the maximum delay in the network. Delay compensation can be very important for high fidelity, noise cancellation, speech recognition, telephony and multi-channel sound applications.

7.7 Node Alive Supervision and Fail Safe Monitoring

The node alive supervision mechanism is used for network management such as channel allocation, error management and power management. In order to avoid channel blocking due to unplugged or defective devices, devices that are detected as „not alive“ can be taken out of(de-allocated from) the resource request list. By this means the freed-up channels previously used will become immediately available to the network.

7.8 Remote Access

Remote access allows for network management functions such as network diagnostics, etc. to be handled in a centralized or decentralized manner within each node, depending on the higher layer software structure.

8 Media and Topology

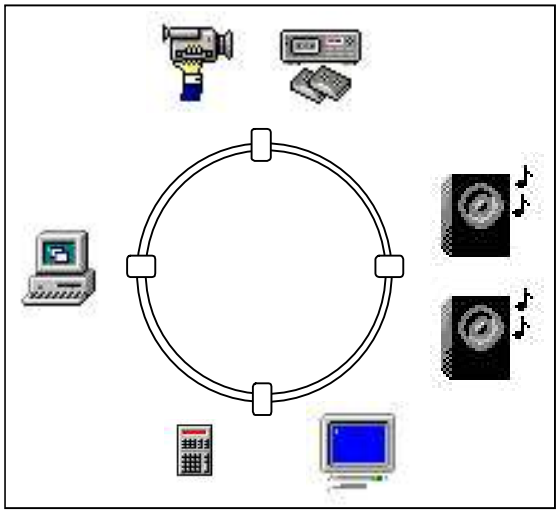
8.1 Physical Wiring Topology

One of the main strengths of the MOST Technology is its flexibility with respect to network topologies. This allows combinations of Ring and Star topologies, providing system designers with the freedom to choose the optimum architecture for any system.

8.1.1 Point to Point Link: Unidirectional or Bi-directional

The one-way point-to-point link is the simplest way of interconnecting devices. This approach can be used only if data transfer is required in a single direction. For example, this would be the case for connecting an audio source to a speaker. In this configuration no information flow is required from the speaker to the audio source. However, this basic configuration can easily be extended to more complex structures, such as branches or bi-directional links. Source (digitized multimedia) and control data are available at the sink, so that control of sinks is possible. One popular method of making similar connections is known as the S/PDIF link, and is often used as an interconnection between audio amplifiers, DAT tape players, Minidisks, CD players and active speakers. MOST is fully compatible to this standard, but provides up to 12 S/PDIF channels concurrently with a control channel over a single fiber.

8.1.2 Ring Topology



MOST can run very efficiently on a ring topology. Advantages of a ring topology are many and include the following:

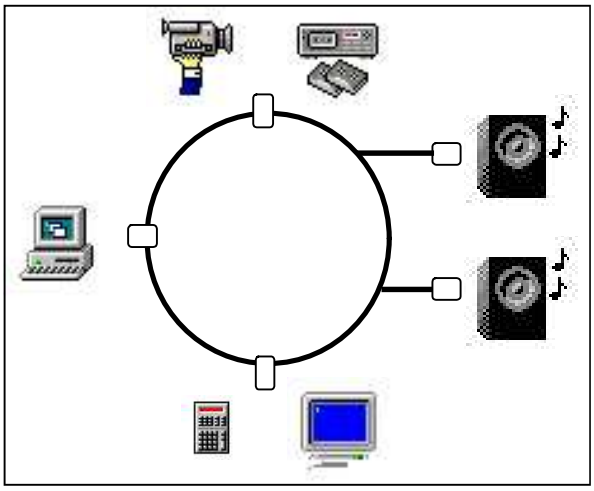
Low overall cost and constant cost per node since there is no overhead cost in the form of a hub, switch, or other shared network resource
Minimum use of physical layer media, reducing system cost and weight
Ease of expansion and no change in the basic architecture (such as the wiring infrastructure) is required
Availability of all source data (e.g. digitized audio) at each device

The ring topology has historically had one major disadvantage associated with it that can be largely overcome in a MOST Network: network reliability. In a traditional ring topology the failure of one node can cause the entire network to fail.

The MOST Network addresses this issue in several ways. First, with respect to system reliability, every MOST transceiver has a bypass mode which can be used when the node is in low power mode (described above). Many failure mechanisms of a node such as power loss, or loss of lock will result in the transceiver going into bypass mode such that the rest of the Network is unaffected. There are still a few cases where a failure at or near a node will affect the entire network, e.g., a cut fiber. In such cases, the damaged node or fiber link must be repaired or replaced to get the network back up and running. This situation can be avoided entirely, albeit at some added expense, by implementing a second, redundant ring. In any case, if the nodes in the network or the connections between them are not easily accessible, and ease of network diagnostics and maintenance is key, the star-adapted ring topology is another possible alternative. In this configuration, every inter-node link is accessible from a single point for diagnostic and bypass purposes.

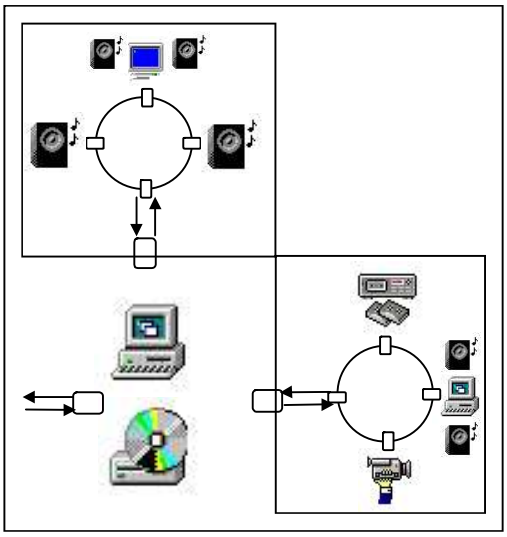
8.1.3 Rings Incorporating Splitters

One-way point-to-point connections can also be incorporated in a Ring by incorporating passive or active optical splitters in a plastic optical fiber network, or simple branches in a network with a copper physical layer.
This would typically be applied in connecting to display and/or active speaker devices that do not have to communicate back to the entire network. One major advantage of this implementation is reduced cost, since no transmitter is required. Active speakers are the most favorable candidates for this application .



8.1.4 Star Topology

Another option is a combination of Star and Ring topologies which may be the best choice for large networks. In structures as required for home networking such a configuration with a central server/ router (i.e. set-top boxes) can have multiple branches, with each of them configured as a Ring structure. This offers the most flexible way of implementing complex structures with easy fault detection and high bandwidth.
Any of the above mentioned topologies or combination of topologies is possible, which offers maximum flexibility in terms of network implementation, depending on the particular situation's requirements. Migration from a simple two node implementation to more complex structures is an easy task without the need of any basic changes.



8.2 Sockets

For applications of MOST technology requiring sockets for the connection of equipment to the network, e.g. home networking, the standard plug can be either electrical or optical. The appropriate solution depends on the implementation and available infrastructure. In the case of an electrical plug which can include A/D and D/A converters for audio, the MOST transceiver is embedded as part of a socket. However, consumer devices, such as active speakers, media players and controllers can also have optical interfaces, which interface directly to the optical network.

8.3 Media

As mentioned above, the ideal physical medium for MOST networks is Plastic Optical Fiber. Even though other media based on copper, like UTP (unshielded twisted pair) or Coax, can be applied, POF offers a number of valuable advantages without any burden in cost. The medium is immune to electromagnetic interference (EMI) and short circuits are not an issue. It can be integrated into the power-line infrastructure with little concern for maximum inter-node distance limitations or any fear of damage to the wiring. If insertion loss is a concern due to distances between nodes of more than 100 meters, the system can be implemented using Hard Clad Silica Fiber (HCSF) which supports a distance between nodes of 100 to 250 meters without any changes in the connectors or elsewhere in the system.

8.4 POF Cables and Connectors

Standard Plastic Optical Fiber cables of 1 mm diameter type commonly used for consumer audio and many other applications are suitable for MOST applications as well. This type of POF cable is readily available from vendors such as Mitsubishi, Toshiba, Torray, and others. Many other common connectors and cables are suitable for use in a MOST Network as well.
A number of highly reliable, inexpensive connectors have been developed for MOST applications by vendors such as HP, Framatome, AMP, Toshiba and others. LED's and Optical Receivers are available with speed and power optimized to the MOST specifications.

The Simplex JIS F05 POF connector for digital audio equipment is one of the most popular low cost POF connectors. For duplex connectors the JIS F07 POF connector is well accepted. These inexpensive connectors are cost competitive with the RJ-45 Modular Jack which is widely utilized for UTP applications. Easy field and automated cable termination capability in the plant is their main feature. The insertion loss is less than 1.5 dB for a 980/1000 POF to POF connection. MOST technology specifies 20 dB dynamic headroom for optical links. This provides enough power budget for a 24Mbit/s link up to 100 m with various POF to POF connectors in between.

9 MOST Application Areas

MOST (Media Oriented Systems Transport) network technology is a versatile, high performance, and low cost multimedia network technology based on synchronous data communication. Ideal for real-time applications such as CD quality audio, surround sound, and high-quality video, it also supports control data and burst-type data transfers for both latency sensitive and latency insensitive network devices.

MOST technology employs a synchronous approach in order to provide a low overhead, low cost network interface to even the most simple multimedia devices. It supports such devices as analog/digital converters for microphones and digital/analog converters for speakers with low intelligence and no buffering capability without any compromise in signal quality. At the same time it provides more complex DSP-based devices with all of the control and multimedia information necessary to fully utilize their capabilities, maximizing the flexibility of the overall system. Equipment such as multimedia computers, analog audio gateways, multimedia CD players, hi-fi audio equipment, telecommunication terminals, video players, TV sets, satellite receivers, set top boxes, etc. can all be networked to interact at the lowest possible cost.

Since most equipment interconnected with this kind of network is capable of digital signal processing, the network is structured in a way that these applications can be implemented very efficiently. Multimedia-related features, such as bandwidth allocation and peer-to-peer communication make it ideal for a serial multimedia consumer network, without the need of a central host. The current version of MOST technology can support over 12 uncompressed, stereo CD-quality audio channels at the same time, or 12 MPEG1 video + audio channels, or several MPEG2 video + audio channels (depending upon the MPEG2 implementation), all with additional bandwidth for control, communication and asynchronous applications.

MOST technology complies with all requirements for different physical layers and is able to support complex topologies required for plastic optical fiber applications or infrared controlled systems. Since plastic fiber is the most promising medium for future home and multimedia networks, the MOST network is future proof and state of the art. Operation over the substantial node to node distances required for multimedia home networking is not an issue using plastic fiber and high speed LED's available today. The system is competitive in cost today compared to copper solutions proposed for similar applications, and has far more cost reduction potential over time. The MOST network can be configured in ring, star, tree or combined topologies and has a flexible 'plug and play' structure.

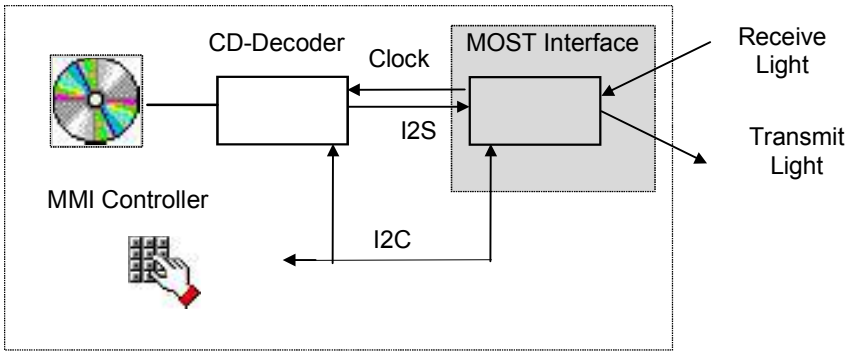
A MOST network can be used in conjunction with a number of different protocols. Since all basic methods required for source and control data communication are provided, the system is very flexible in terms of compatibility with a number of protocol layers.

A MOST Network is the most cost-effective, high performance solution available today for any environment where the primary data types to be networked are multimedia (digitized audio, voice, and compressed video) and control. It is not designed to replace traditional LAN's supporting client/server applications and is focused on multimedia services required for home and consumer networks rather than on asynchronous file transfer. While attempts have been made to shoehorn multimedia functions into traditional asynchronous LANs, e.g. ISO-Ethernet, PACE, etc., the multimedia capability they provide is quite limited and is still secondary to the primary function of the network, file transfer or line optimization. On the other hand, MOST networks can handle file transfer services and peer to peer computer networking efficiently as long as this is not the primary function of the network.

9.1 Consumer Electronics

The simplicity of the MOST interface makes it ideally suited for consumer multimedia products. For the many consumer product manufacturers who have already adopted the S/PDIF interface to interconnect audio equipment, MOST technology provides an ideal migration path toward real multimedia networking. Its structure is very similar to S/PDIF and it can be implemented at a comparable cost. For example, the following block diagram shows an implementation of the MOST interface in a multimedia CD player.

Block diagram of a CD player with MOST Interface



9.2 Multimedia Computers

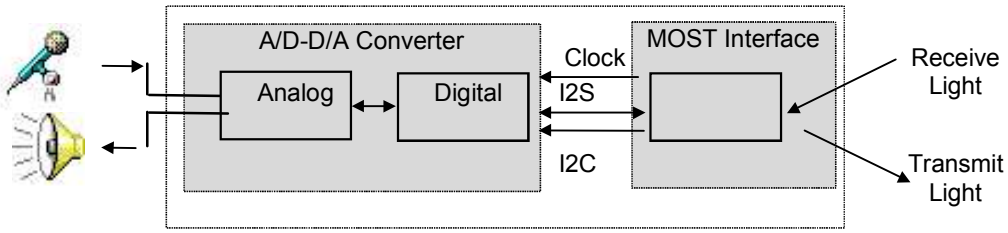
The multimedia computer has already become an integral part of the home consumer electronics environment. As such, a low cost interface to the home network will be required in the near future, which takes into account the structure of consumer electronic devices. It can also be envisaged that multimedia computers will be the servers for home automation and multimedia services. No existing networking solution addresses this need in a practical way. Since a MOST interface can be embedded into a computer platform for little more than the cost of the physical connection, it provides a robust, cost-effective interface from the PC to remote home multimedia and control applications. In addition, with a POF physical layer, distance between nodes (up to 250 m), EMI (electromagnetic interference) and ground loop hum for audio applications are not issues as they are with other solutions utilizing electrical media.

9.3 Home Multimedia Networking

A number of standards proposals are in the works for home multimedia networking. However, no real standard currently exists. A number of control-oriented home networks are already widely established using such technologies as LonWorks, and CEbus. However, satisfying the requirements of an information-oriented society (e.g. multi-room audio and video, and hands-free telecommunications from room to room or from room to the outside world) is a strong argument for a low cost in-house audio/video and communication solution as well. In fact, it is clear that control and multimedia networking in the home will converge in the future, and that a low cost solution will be required. This solution must also be able to make use of the centralized intelligence provided by a multimedia server, in order to have access to such services as speech recognition and voice control of the consumer devices on the network.

The availability of audio and video, coming from a central media server will become central to future lifestyles. MOST Technology provides all basics to implement such systems now, without the need of complex computing and intelligence within the terminals. The following example of a standalone analog audio gateway shows the simplicity of such implementations.

Block diagram of an Analog Audio Gateway with MOST Interface



A simple codec function without any glue logic can interface directly to the MOST Core. The output bit streams of many A/D converters can be directly connected to the MOST source data ports. Even remote control of the codec is possible through the network without the need for any local micro-controller.

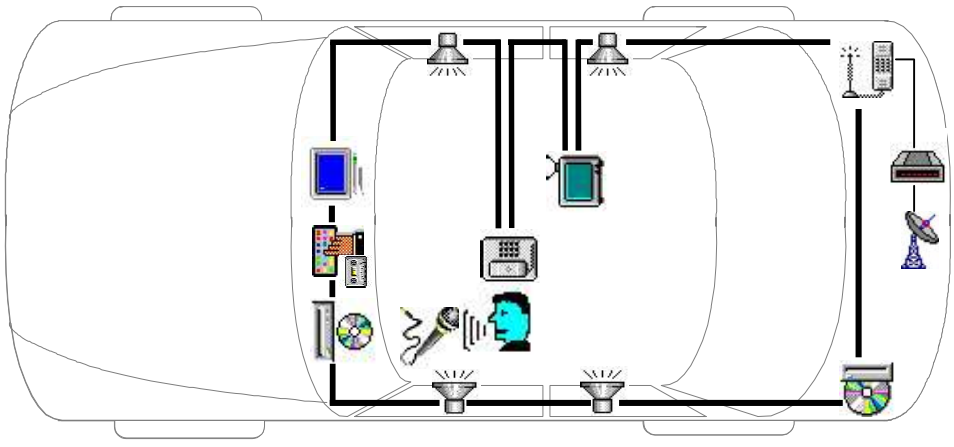
9.4 Automotive Multimedia Networking

The need for multimedia networking in cars has been much more obvious than anywhere else. Speech recognition provides the only way for the driver to operate complex equipment such as telecommunication and navigation devices without significant safety concerns. The automotive industry has already adopted the MOST Technology as an integral part of their future strategy.

Reliability of the system and cost effectiveness were major considerations in the decision making process. In addition, the technology was adopted due to its simplicity, data capacity and immunity to EMI.

The optical system shown in the following diagram reduces the weight of the original copper wire system by 4.5 kg and saves more than 250 m of cable. The resulting flexibility is much higher as well. A simple system can be expanded without changing the architecture and there is no additional cost per node for a system with fewer nodes. All devices shown can be voice controlled without any additional wiring. The cost savings for such a system versus a traditional copper implementation is significant.

Block Diagram of a Car Multimedia System



Voice activated Navigation, Communication and Entertainment System

10 Cost Considerations

The cost of a multimedia network is very difficult to define, since there are a lot of hidden costs such as cable installation, trouble shooting and compliance with federal EMC/ EMI requirements. However, the cost floor is determined by the per node implementation cost, in particular for devices with low intelligence and bandwidth requirements. As a consequence, control-only devices and voice interfaces need to be integrated without a major burden in terms of complexity and cost. MOST Technology elegantly addresses this requirement in particular.

It should also be noted that once a home network is installed with a particular maximum bandwidth limitation, there is no cost savings for unused bandwidth. In other words, a 10Mbps connection costs the user the same as a 1Mbps connection would. As a consequence, statistical line optimization techniques (e.g., as used in ATM) are not cost effective in a home network since higher per node costs are involved.

10.1 IC Cost

As the IC-related cost of the system is proportional to the number of nodes within the network, overkill must be avoided in supporting simple network equipment. Since audio is one of the most critical applications in terms of cost sensitivity and quality, it is instructive to see how MOST technology has been designed to provide an ultra low cost connection to an audio device. First of all, there is no need for memory to buffer the audio data at the node due to the synchronous nature of the data link. There is also no need for any local computation capability for network administration. The typical IC cost for a simple analog audio gateway is expected to be only about 5 dollars.

10.2 Cable Cost

The cost for POF (plastic optical fiber) is very competitive with the cost of copper-based wiring. The cost of POF cable today is less than $0.30 U.S. per meter, even lower than high quality Coax. Also, POF cable is relatively new and is expected to become significantly less expensive over time. In contrast, copper-based media have already gone through the manufacturing “learning curve” and have little room for future cost reduction.

10.3 Terminal Cost

The cost of terminating POF at network nodes is much lower than for glass fiber. Since the termination of POF is an easy task and is not even required until a terminal is needed (the fiber can be left not terminated in the wall), there is little cost to the builder of a new home to make it Home Network-ready. When a terminal or home network socket is installed, the material cost primarily consists of the optical receiver and transmitter devices (~$2 U.S. for a set).

10.4 System Cost and Flexibility

The system cost is low, since there is no need for additional circuitry in any device beyond the optical transducers and the interface IC. The network management is embedded into the network transceiver and therefore adds nothing to the system cost. Even very simple nodes (i.e., without a microcontroller) can be realized, which are remotely controlled by any other intelligence within the network. On the other hand, the flexibility of the network is very high, since all kinds of network standards can be easily interfaced to the system. This includes control networks and multimedia networks at the same time.

As the bandwidth of MOST technology increases over time through the utilization of higher speed LED's, higher speed C-MOS devices, etc., the system speed will increase as well. However, the same infrastructure should be appropriate as for today's technology, and no changes should be required to internal device interfaces or software structures.

11 Interface to other Systems

11.1 Direct Serial, Real-time, PCI, ISA or Serial Control Bus Implementations

The MOST Network can interface with a number of different external data busses, including PCI, ISA, RS 232, I^2C, SPI, I^2S, S/PDIF and others. The choice of the appropriate interface depends on the requirements of the application in terms of data transfer, data characteristics (i.e. latency sensitive or not, burst, continuous, ...) and on the overall structure of the device (i.e. Computer, CD device, A/D - D/A converter, etc.).

11.2 MOST Core and other System Solutions

The MOST core as used today provides all basic structures to interface with any of the above mentioned data busses. Different I/O modules are available as well to build the appropriate physical interface layer for each application.

12 Interface to other Network Standards

This section gives some examples of ways to interface MOST Technology with many of today's other multimedia networking standards and proposals . Since MOST is a technology rather than a networking standard it provides multiple approaches to interface with other systems. Its flexibility in this respect is an important and distinctive feature, since it provides system designers a number of basic mechanisms that give access to a fiber optic physical layer (i.e. TDM, Token Ring, CSMA).

12.1 Interface to AES/ EBU - S/PDIF

All MOST IC's provide a direct interface to AES/EBU and S/PDIF as a standard interface. This bit stream can be transported on the MOST network transparently. In fact, multiple S/PDIF channels can easily be transported by the network, together with any associated control information. One convenient control layer in that field is provided by the AES 24 specification making it even easier to upgrade and/or cost-reduce systems employing current multiple channel AES/EBU transmitter and receiver and control technology.

The MOST network is also flexible in the sense that input and output interfaces for source data don't have to be of the same type. For example, source data can be input to a network node through an I^2S (serial) or parallel interface and output at a different node through an S/PDIF interface, or vice versa.

As such MOST is an ideal solution for networking of musical instruments, digital audio processing equipment, digital mixing consoles and post-processing devices.

12.2 Interface to other Control Networks

The structure of the MOST Control Frame offers flexible access to the control data communication function of the MOST bit stream. The availability of singlecast and broadcast mechanisms as well as fairness arbitration combined with high speed data transfer guarantees maximum compatibility and system integration capability.

Source data ports can also be used as high speed asynchronous point to point connections (e.g., ISDN) as long as reasonable oversampling is provided. RS 232 links are available by making use of this feature.

Anything from simple controls to multimedia inputs and outputs are easily remotely controllable via the network. With almost no additional overhead to the system, sensors for applications such as security systems can be included at the nodes as well as microphones and speakers.

13 System Simulation

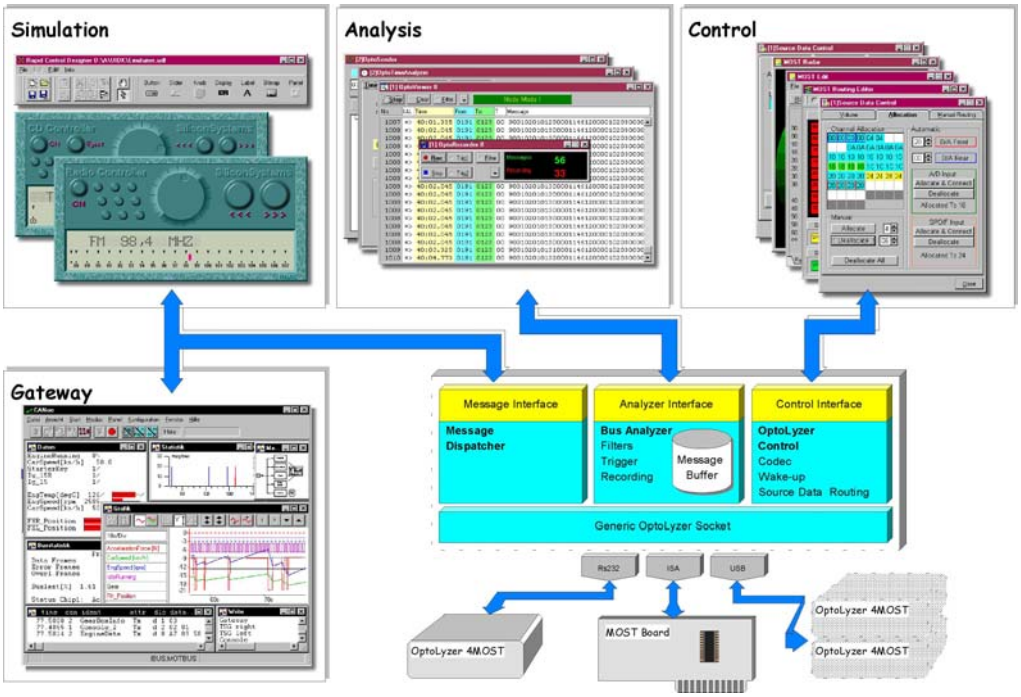
When developing applications for MOST devices, simulating the application in a virtual MOST environment helps to optimize the development process. A simulation can be used for testing before the respective hardware is available. The optimal approach for simulation is, to combine simulation of virtual MOST devices with devices of the real world, like it is implemented in OptoLyzer*4*MOST® Professional.

OptoLyzer*4*MOST® Professional consists of a collection of library functions which cover all aspects of analysis and simulation of a MOSTNetwork. OptoLyzer® Socket Library has a generic interface for the hardware. Adapting to different OptoLyzer hardware components is done by device-specific libraries (DLLs), which can be integrated into the system subsequently (e.g. together with a new hardware).

The structure of the OptoLyzer® Socket Library is based on three main functional blocks:

- OptoLyzer Control Block
- Bus Analyzer
- Message Dispatcher

The figure below shows the basic structure of the OptoLyzer® Socket Library:



The simulation function of OptoLyzer*4*MOST® Professional covers all steps of development of a MOST system. It is possible to simulate virtual MOSTDevices communicating with other virtual MOSTDevices in a virtual MOSTNetwork. In addition to that, it is possible to mix real and virtual MOSTDevices, so that virtual MOSTDevices can communicate with real devices.

It is the Message Dispatcher that links the virtual and the real MOST Network. It provides a message interface, which can support almost any number of virtual MOST Devices. The interface itself consists of functions that provide sending and receiving of MOST messages and the handling of group address and logical address. For a simulated MOST Device, the OptoLyzer® Socket Library behaves like a real MOST node. In addition to that, a real MOST Transceiver can be assigned to a virtual MOST Device in an OptoLyzer*4*MOST® Interface Box or a PC board. The simulated MOST Devices will then get „visible“ and addressable for real MOST Devices.

The possibility to simulate virtual devices in a virtual network based on OptoLyzer® Socket Library, allows to test the interaction between MOST Devices without having them developed (yet).

14 Terminology

The following provides a list of terms and abbreviations used within this specification.

ACK	Acknowledgment. Handshake packet indicating a positive acknowledgment.
Active Device	A device that is powered and not in the suspend state.
API	Application Programming Interface.
Application Socket	See „MOSTNetServices Application Socket“.
Arbitration	Management of access to the network.
Asynchronous Data	Data transferred at irregular intervals with relaxed latency requirements.
Asynchronous SRC (sample rate conversion)	The incoming sample rate, F_{S_i}, and the outgoing data rate F_{S_0}, of the SRC process are independent (i. e., no shared master clock).
Audio Device	A device that sources or sinks sampled analog data.
Bandwidth	The amount of data transmitted per unit of time, typically bits per second (bps) or bytes per second (Bps).
Basic Layer System Services	See „MOSTNetServices Basic Layer“.
Bit	A unit of information used by digital computers. Represents the smallest piece of addressable memory within a computer. A bit expresses the choice between two possibilities and is typically represented by a logical one (1) or zero (0).
bps	Transmission rate expressed in bits per second.
Broadcast address	Network address for all network devices.
Broadcasting	Transferring data to all network devices.
Buffer	Storage used to compensate for a difference in data rates or time of occurrence of events, when transmitting data from one device to another.
Bulk Transfer	Non periodic, large burst-like communication typically used for a transfer that can use any available bandwidth and also be delayed until bandwidth is available.
Bus Configuration	Detecting and identifying MOST devices.
Byte	A data element that is eight bits in size.
Central Registry	Contains a lookup-table for cross-referencing physical and functional addresses. Implemented in the Network Master.

Channel allocation	A method of allocating synchronous bandwidth on the network.
Channel cluster	A number of logical channels grouped together in an application layer, such as multi-channel audio etc.
channel de-allocation	A method of de-allocating synchronous bandwidth on the network.
channel ID	Identifier of a logical channel in the network.
Characteristics	Those qualities of a network device that are unchangeable; for example, the device class is device characteristic.
Client	Software resident on the host that interacts with host software to arrange data transfer between a function and the host. The client is often the data provider and consumer for transferred data.
COM Port	Communication port. On personal computers, an eight-bit asynchronous serial port is typically used.
Configuring Software	The host software responsible for configuring a network device. This may be a system configurator or software specific to the device.
Control Data	One of three data types in a MOST system. These data consist of three types of Control transfer support: configuration/ command/ and status/ data governing communications between devices and functions.
Control Data channel	A dedicated asynchronous channel for system control, remote control and control status communications.
Control Pipe	See „message pipe“
CRC (Cyclical Redundancy Check)	Cyclical Redundancy Check - a check performed on data to see if an error has occurred in transmitting, reading or writing data. The result of a CRC is stored or transmitted with the checked data. The stored or transmitted result is compared to a CRC calculated for the data to determine if an error has occurred.
De-Central Registry	Contains a lookup-table for cross-referencing physical and functional addresses. Implemented in each node.
Default Address	An address defined by the MOST Specification and used by a network device when it is first powered-up or reset. The default address is 0x0FFF.

Device	A logical or physical entity that performs a function. The actual entity described depends on the context of the reference. At the lowest level, „device“ may refer to a single hardware component, such as a memory device. At a higher level, it may refer to a collection of hardware components that perform a particular function, such as a MOST interface device. At an even higher level, device may refer to the function performed by an entity attached to the Universal Serial Bus; for example, a data/ FAX modem device. Devices may be physical, electrical, addressable, and logical.
Device Address	The address of a device on a MOST network. The Device Address is the Default Address when the network device is first powered or reset. Devices are assigned a unique Device Address by the MOST System software after network initialization. (See „Device Unique Address“ below)
Device unique address	The unique device address is determined by the physical position of the device within the network, relative to the frame generator.
Downstream	The direction of data flow from the frame generator or away from the frame generator. Downstream ports receive upstream data traffic.
Driver	When referring to hardware, an I/O pad that drives an external load. When referring to software, a program responsible for interfacing to a hardware device; such as a device driver.
DWORD	Double word. A data element that is 2 words, 4 bytes, or 32 bits in size.
Dynamic Insertion and Removal	The ability to attach and remove devices while the host is in operation. Also known as „hot plug-in.“
E^2PROM	See EEPROM.
EEPROM	Electrically Erasable Programmable Read Only Memory. Non-volatile reusable memory storage technology.
End User	The user of a system.
EOP	End of packet.
Frame	The time from the start of one Preamble to the start of the subsequent Preamble; consists of a series of transactions.
Frame generator	A single device within the network providing the timing for the network.
Fs	Sampling Frequency: See Sample Rate.
Full-duplex	Data transmission occurring in both directions simultaneously.
Function	A device that provides a capability to the network. For example, an ISDN connection, a digital microphone, or speakers.

Functional Addressing	Addressing by Function Block and Instance ID.
FBlock	Unit of functions belonging together, e.g. CD player
Function Block	See „FBlock“
Handshake Packet	A packet that acknowledges or rejects a specific condition. For examples, see ACK and NACK.
Host	The host computer system where the host controller is installed. This includes the host hardware platform (CPU, bus, etc.) and the operation system in use.
Host Controller	The host's Network interface.
Host Controller Driver	The host software layer that abstracts the host controller hardware. The Host Controller Driver provides an SPI (System Programming Interface) for interaction with a host controller. The Host Controller Driver hides the specifics of the host controller hardware implementation.
Host Resources	Resources provided by the host, such as buffer space and interrupts.
Hub	A device that provides additional managed or unmanaged connections to the Network.
I^2C	Acronym for the Inter-Integrated Circuits serial interface. The I^2C interface was invented by Philips Semiconductors.
Instance ID (InstID)	Identifier for differing identical function blocks in a MOST network.
Industry Standard Architecture (ISA)	The 8 and/ or 16 bit expansion bus for IBM AT or XT compatible computers.
Initiator	See Token Ring Generator
Integrated Services Data Network (ISDN)	An internationally accepted standard for voice, data, and signaling using public, switched telephone networks. All transmissions are digital from end-to-end. Includes a standard for out-of-band signaling and delivers significantly higher bandwidth than POTS. Basic rate ISDN provides two 64kbps bearer channels and one D channel for control data.
Interrupt Request	A hardware signal that allows a device to request attention from a host. The host typically invokes an interrupt service routine to handle the condition which caused the request.
IRQ	See Interrupt Request.
ISA	See Industry Standard Architecture.
ISDN	See Integrated Services Data Network.
Isochronous Data	A stream of data whose timing is implied by its delivery rate.
Isochronous Transfer	Isochronous transfers are used when working with isochronous data. Isochronous transfers provide periodic, continuous communication between host and device.

Jitter	A tendency toward lack of synchronization caused by mechanical or electrical changes. More specifically, the phase shift of digital pulses over a transmission medium.
kbps	Transmission rate expressed in kilobits per second.
kBps	Transmission rate expressed in kilobytes per second.
Line Printer Port	A port used to access a printer. On most personal computers an eight-bit parallel interface is used.
Logical address	Address of a node, that can be defined by application. Must be unique!
Logical channel	A number of bytes grouped together under a particular channel ID and available within one channel allocation request, such as a 8/16/24/32 bit audio channel.
Low Level System Services	These services provide fundamental functions from physical interfacing up to transaction layer.
LPT Port	See Line Printer Port.
LSB	Least Significant Bit.
MBaud	Transmission rate expressed in megabits per second.
Mbps	Transmission rate expressed in megabits per second.
MBps	Transmission rate expressed in megabytes per second.
Message Pipe	A pipe that transfers data using a request/ data/ status paradigm. The data has an imposed structure which allows requests to be reliably identified and communicated.
Modem	An acronym for Modular/ Demodulator. Component that converts signals between analog and digital. Typically used to send digital information from a computer over telephone network which is usually analog.
MOST	Media Oriented Systems Transport - the method of data transportation in a MOST system, in particular the architecture and management of the bit stream.
MOST Transceiver	Chip in which the Low Level System Services are implemented (e.g. OS 8104 of Oasis SiliconSystems).
MOSTNetServices Application Socket	Programmer's Library in ANSI C. Forms the basic core of the application area.
MOSTNetServices Basic Layer	Programmer's Library in ANSI C. Provides access on the Low Level System Services.
MSB	Most Significant Bit.
NACK	Negative Acknowledgment. Handshake packet indicating a negative acknowledgment.

Network Master	Optional central instance in a MOST network, that contains e.g. the central registry.
Non Return to Zero Invert (NRZI)	A method of encoding serial data in which ones and zeros are represented by opposite and alternating high and low voltages where there is not return to zero (reference) voltage between encoded bits. Eliminates the need for clock pulses.
NRZI	See Non Return to Zero Invert.
Object	Host software or data structure representing a Network entity.
Packet	A bundle of data organized in a group for transmission. Packets typically contain three elements: 1) control information (e.g., source, destination, and length), 2) the data to be transferred, and 3) error detection and correction bits.
Packet Buffer	The logical buffer used by a network device for sending or receiving a single packet. This determines the maximum packet size the device can send or receive.
PBX	See Private Branch eXchange.
PCI	See Peripheral Component Interconnect.
PCMCIA	See Personal Computer Memory Card Industry Association.
Peripheral Component Interconnect	A 32- or 64-bit, processor independent expansion bus used on personal computers.
Personal Computer Memory Card International Association	The organization that standardizes and promotes PC Card technology.
Physical channel	A single byte in a synchronous transfer channel.
Physical device	A device that has a physical implementation such as speakers, microphones, CD players, tuners, telephone etc.
Pipe	A logical abstraction representing the association between an endpoint of a device and software on the host. A pipe has several attributes; for example, a pipe may transfer data as streams (Stream Pipe) or messages (Message Pipe).
Plain Old Telephone Service (POTS)	Basic access to the PSTN (public switched telephone network) via analog telephony using twisted pair.
PLL	Phase Locked Loop. A circuit that acts as a phase detector to keep an oscillator in phase with an incoming frequency.
Plug and Play (PnP)	A technology for configuring I/O devices to use non-conflicting resources in a host. Resources managed by Plug and Play include I/O address ranges, memory address ranges, IRQs, and DMA (direct memory access) channels.
PnP	See „Plug and Play“.

POF	Plastic Optical Fiber
Polling	Asking multiple devices, one at a time, if they have any data to transmit.
POR	See Power On Reset.
Port	Point of access to or from a system or circuit.
POTS	See Plain Old Telephone Service.
Power On Reset (POR)	Restoring a storage device, register, or memory to a predetermined state when power is applied.
Programmable Data Rate	Either a selectable fixed data rate (single frequency endpoints), a limited number of data rates (32kHz, 44.1kHz, 48kHz, ...), or a continuously programmable data rate. The exact programming capabilities of an endpoint must be reported in the appropriate class-specific endpoint descriptors.
Protocol	A specific set of rules, procedures, or conventions relating to format and timing of data transmissions between two devices.
RA	See Rate Adaptation.
Rate Adaptation	The process by which an incoming data stream, sampled at FS_i is converted to an outgoing data stream, sampled at FS_0 with a certain loss of quality, determined by the rate adaptation algorithm. Error control mechanisms are required for the process. FS_i and FS_0 can be different and asynchronous. FS_i is the input data rate of the RA; FS_0 is the output data rate of the rate adapter.
Request	A request made to a network device contained within the data portion of a packet.
Routing Engine	Routing processor dedicated to bit stream routing.
Sample	The smallest unit of data on which an endpoint operates; a property of an endpoint.
Sample Rate (FS**)**	The number of samples per second, expressed in Hertz.
Sample Rate Conversion (SRC)	A dedicated implementation of the RA process for use on sampled analog data streams. The error control mechanism is replaced by interpolating techniques.
SCSI	See Small Computer Systems Interface.
Service	A procedure provided by an SPI.
Service Interval	The period between consecutive requests to a network node to send or receive data.
Service Rate	The number of services to a given endpoint per unit in time.
Single-casting	Message for one single node in the network.

Small Computer Systems Interface (SCSI)	A local I/O bus that allows peripherals to be attached to a host using generic system hardware and software.
Source Data	Data transported as real - time data, or asynchronous data.
Source Data channel	A synchronous channel containing a real-time data stream.
S/PDIF	Sony Philips Digital Interface for digital audio transmission.
SPI	See System Programming Interface.
SRC	See Sample Rate Conversion.
Stage	One part of the sequence composing a control transfer; i. e., the setup stage, the data stage, and the status stage.
Synchronous RA	The incoming data rate, Fs_i , and the outgoing data rate, Fs_0, of the RA process are derived from the same master clock. There is a fixed relation between Fs_i and Fs_0.
Synchronous SRC	The incoming data rate, Fs_i , and the outgoing data rate, Fs_0, of the SRC process are derived from the same master clock. There is a fixed relation between Fs_i and Fs_0.
System Programming Interface (SPI)	A defined interface to services provided by system software.
TDM	See Time Division Multiplexing.
Termination	Passive components attached at the end of cables to prevent signals from being reflected or echoed.
Time Division Multiplexing (TDM)	A method of transmitting multiple signals (data, voice, and/ or video) simultaneously over one communication medium by interleaving a piece of each signal one after another.
Time-out	The detection of a lack of bus activity for some predetermined interval.
Timing master	Same as frame generator.
Timing slave	A device slaving its timing to the bit stream with a PLL.
Token Generator	Generates the tokens to nodes on a token ring. A node which seizes a token is then able to send data around the network.
Token Packet	A type of packet that identifies what transaction is to be performed on the bus.
Transaction	The delivery of service to an endpoint; consists of a token packet, optional data packet, and optional handshake packet. Specific packets are allowed/ required based on the transaction type.
Transfer	One or more bus transactions to move information between a software client and its function.

Transfer Type	Determines the characteristics of the data flow between a software client and its function. Four Transfer types are defined: Control, interrupt, bulk, and isochronous.
Universal Serial Bus (USB)	A collection of Universal Serial Bus devices and the software and hardware combination that allows them to connect their capabilities and functions to the host.
Universal Serial Bus Interface	The hardware interface between the Universal Serial Bus cable and a Universal Serial Bus device. This includes the protocol engine required for all Universal Serial Bus devices to be able to receive and send packets.
Upstream	The direction of data flow towards the host. Upstream ports receive downstream data traffic.
USB	See Universal Serial Bus.
Virtual Device	A device that is represented by a software interface layer; e.g., a hard disk with its associated device driver and client software that makes it able to reproduce an audio „.WAV“ file.
Word	A data element that is two bytes or 16 bits in size.

Modifications without written notice

Annex 2

MOST

Media **O**riented **S**ystems **T**ransport

Multimedia and Control Networking Technology

MOST Specification
Rev 2.1
02/2001

Version 2.1-00

Legal Notice

COPYRIGHT

LICENSE DISCLAIMER

Nothing on any MOST Cooperation Web Site, or in any MOST Cooperation document, shall be construed as conferring any license under any of the MOST Cooperation or its members or any third party's intellectual property rights, whether by estoppel, implication, or otherwise.

CONTENT AND LIABILITY DISCLAIMER

MOST Cooperation or its members shall not be responsible for any errors or omissions contained at any MOST Cooperation Web Site, or in any MOST Cooperation document, and reserves the right to make changes without notice. Accordingly, all MOST Cooperation and third party information is provided "AS IS". In addition, MOST Cooperation or its members are not responsible for the content of any other Web Site linked to any MOST Cooperation Web Site. Links are provided as Internet navigation tools only.

MOST COOPERATION AND ITS MEMBERS DISCLAIM ALL WARRANTIES WITH REGARD TO THE INFORMATION (INCLUDING ANY SOFTWARE) PROVIDED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

In no event shall MOST Cooperation or its members be liable for any damages whatsoever, and in particular MOST Cooperation or its members shall not be liable for special, indirect, consequential, or incidental damages, or damages for lost profits, loss of revenue, or loss of use, arising out of or related to any MOST Cooperation Web Site, any MOST Cooperation document, or the information contained in it, whether such damages arise in contract, negligence, tort, under statute, in equity, at law or otherwise.

FEEDBACK INFORMATION

Any information provided to MOST Cooperation in connection with any MOST Cooperation Web Site, or any MOST Cooperation document, shall be provided by the submitter and received by MOST Cooperation on a non-confidential basis. MOST Cooperation shall be free to use such information on an unrestricted basis.

TRADEMARKS

MOST Cooperation and its members prohibit the unauthorized use of any of their trademarks. MOST Cooperation specifically prohibits the use of the MOST Cooperation LOGO unless the use is approved by the Steering Committee of MOST Cooperation.

SUPPORT AND FURTHER INFORMATION

For more information on the MOST technology, please contact:

MOST Cooperation
Administration
P. O. Box 4327
D-76028 Karlsruhe
Germany

Tel: (+49) (0) 721 966 50 00
Fax:(+49) (0) 721 966 50 01
E-mail: contact@mostcooperation.com
Web: www.mostcooperation.com

Contents

Bibliography

Number	Document
[1]	MOST Specification Framework
[2]	MOST Specification
[3]	MOST High Protocol Specification
[4]	MOST NetServices “Basic Layer”; User Manual and Specification
[5]	MOST NetServices “Application Socket”; User Manual and Specification
[6]	FOT Datasheet
[7]	MOST Transceiver Datasheet
[8]	MOST FunctionCatalog

Document History

Changes MOST Specification 2V0-01 to MOST Specification 2V1-00

Change Ref.	Section	Changes
2V1_001	1	- Added paragraph introducing object oriented approach
2V1_002	2.3.2.2	- FBlockIDs “System Specific” and “Supplier Specific” added (WG-DA 2000-09-12)
2V1_003	2.3.2.4	- FktIDs “System Specific” and “Supplier Specific” added (WG-DA 2000-09-12) - Handling of proprietary Functions/ Function Blocks by controller added (WG-DA 2000-02-09)
2V1_004	2.3.2.2	- Speech output Device added (WG-DA 2000-09-12) - Speech Database Device added (WG-DA 2000-09-12) - Corrected FBlockIDs DAB Tuner (0x43) and TMCTuner (0x41) - FBlock Satellite Radio (0x44) added - FBlock HeadphoneAmplifier (0x23) added - FBlock AuxiliaryInput added (0x24) - FBlock MicrophoneInput added (0x26) - FBlock (0x51) "Telephone mobile" replaced by "Phonebook" - FBlock Router added (0x8) (WG-DA 2001-01-17)
2V1_005	2.3.2.5.1	- Specification of “Error Secondary node” revised - Specification of “Error Device Malfunction” added (WG-DA 2000-05-04) - Specification of “Segmentation Error” added (WG-DA 2000-09-12) - Hint to avoiding “infinite loops” added - "No error replies allowed in case of reception of broadcasted messages" added - Specification of “ Error Method Aborted” added (WG-DA 2000-11-22) - Added remark that methods in general should be aborted only by that application, which has started the method. - Code 0x05 and 0x06: Returning of the value of first incorrect parameter is optional (WG-DA 2001-01-17).
2V1_006	2.3.2.5	- Renamed StartAck -> StartResultAck (0x6) and adapted every occurrence in specification document. - Added AbortAck (0x7) - Added New StartAck (0x8)
2V1_007	2.3.2.5.4	- Added New StartAck (0x8)
2V1_008	2.3.2.5.11	- Added AbortAck (0x7)
2V1_009	2.3.2.6	- Maximum value for LENGTH changed to 65535
2V1_010	2.3.2.7	- Encoding of signed values added - Codes for ISO 8859/15 8 bit and UTF8 added - Maximum value for LENGTH changed to 65535 - Examples enhanced - Data type Boolean revised - Data type BitField added - Description of String enhanced (Null Strings)
2V1_011	2.3.2.5.3	- Flow chart “ Flow for handling communication of methods (controller’s side)”. Error handling for “Timeout = YES” added - Changing of timeout (100ms) for "PROCESSING"
2V1_012	2.3.11.1.2	- Specification of NSteps extended - Units for Speed (m/s), Angle and Pixel added
2V1_013	2.3.11.1.4	- Interpretation of Increment and Decrement added
2V1_014	2.2.6	- Handling of dynamic changes of Function Interfaces through Notification added

Change Ref.	Section	Changes
2V1_015	General	- MMI replaced by HMI (Human Machine Interface)
2V1_016	3.9	- Description of Secondary Node added
2V1_017	3.2.6.8	- Section completely removed, due to an overlapping with the MOST Function Catalog
2V1_018	3.2	- Generally revised
	3.2.2	- Figure 3-3 “Diagnosis Normal Shutdown” changed to “Diagnosis Ready”
	3.2.2.1	- Table 3-6 changed
	3.2.3.1	- “Network Slave” removed, “Requesting System Configuration – Network Master” added
	3.2.3.2	- “Network Master” removed, “Requesting System Configuration – Network Slave” added
2V1_019	3.2.4	- Dynamic Behavior of Secondary Nodes added
2V1_020	3.2.6.4	- "Failure Of A Function Block" added
2V1_021	3.8	- Timeout $t_{Runtime}$ added - Timeout $t_{CfgStatus}$ changed - Timeout t_{Answer} changed - Timeout t_{Diag_Master} changed - Timeout t_{Diag_Slave} changed
2V1_022	2.3.12	- Error handling in case of property failure added - Notification of Function Interface (FI) added (WG-DA 2001-01-17) - Error handling added, in case of values in property not yet available during subscription (WG-DA 2001-01-17)
2V1_023	2.3.2.2	- Note about FBlockID 0xFF added
2V1_024	2.3.11.2.4.2	- Added parameters CurrentSize and AbsolutPosition to description of ArrayWindows - Added PositionTag, and descriptions for PositionTag and WindowSize (WG-DA 2001-01-17)
2V1_025	2.3.2.5.10 2.3.2.5.11	- Added remark that methods in general should be aborted only by that application, which has started the method.
2V1_026	2.3.11.1	- Function Class BoolField added - Function Class BitField added - Description of parameter "OPType" enhanced
2V1_027	3.2.2.4	- Start of Ring Break Diagnosis revised
2V1_028	3.2.5.1	- Note about wakeup methods added
2V1_029	4.6, 4.7	- Voltage levels and Implementation of Power Supply Area are no longer normative.
2V1_030	General	- Eithernet replaced by Ethernet
2V1_031	3.2.2.2	- Behavior of a waking Slave device (Figure 3-6)
2V1_032	3.5.3	- "MOST Asynchronous Medium Access Control (MAMAC)" added
2V1_033	3.4.3.3	- Equation for delay compensation revised ($T_{Source} < T_{Node}$)
2V1_034	4.2.4	- Hint Added. Description of Pig Tail is only one of the possible implementations.
2V1_035	3.4.1.1.2.1	- Method SourceActivity added
2V1_036	3.2.6	- General handling of errors. Synch. connections are removed in case of Fatal Errors.

Changes MOST Specification 2V0-00 to MOST Specification 2V0-01

Change Ref.	Section	Changes
2V01_001	General	Document no longer specified as “Confidential”; Legal Notice inserted.

Changes MOST Specification 1V0 to MOST Specification 2V0

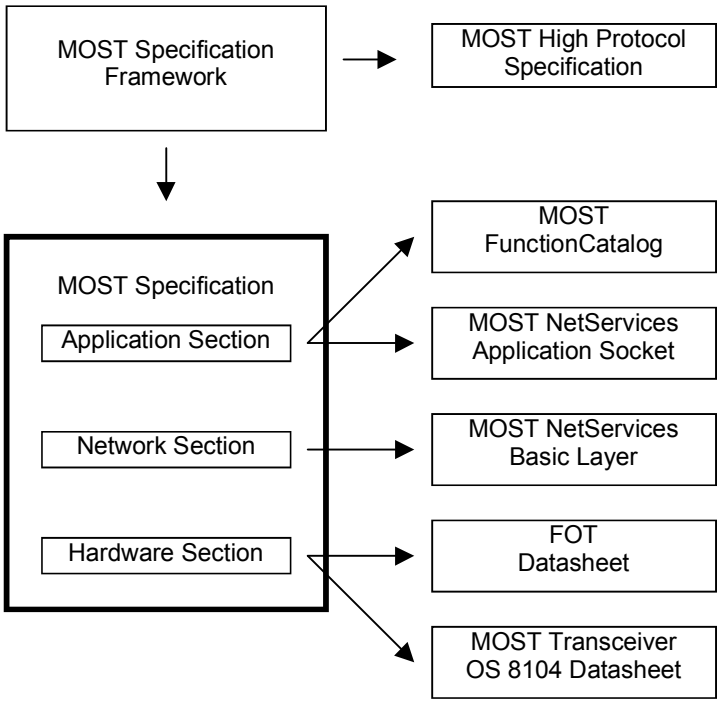
Change Ref.	Section	Changes
2V0_001	3.3.1	Equation modified; Startup address 0xFFFF
2V0_002	2.1.2/ 2.2.5	Section 2.2.5 moved to 2.2.2
2V0_003	2.2.1	NetBlock “functions related to the entire device.”
2V0_004	2.3.2.2	Table 2-5: Proprietary FBlockIDs 0xF0..0xFE
2V0_005	2.3.2.3	Completely revised
2V0_006	2.3.2.4	Minor modification
2V0_007	2.3.2.5	Completely revised
2V0_008	2.3.2.6	Completely revised
2V0_009	2.3.2.7	Bool-field introduced; Definition of STRING expanded, Examples for Exponent, Step and Unit
2V0_010	2.3.5	Minor modification
2V0_011	2.3.6	Distinguishing Properties and Methods; Communication with routing revised
2V0_012	2.3.10	Transmitting function interfaces. Introduced.
2V0_013	2.3.11	Function Classes (completely revised)
2V0_014	2.3.12	Notification for array properties; Notification re-build at system start
2V0_015	3.2.2.2	Error_t_slave replaced by Error_NSInit_Timeout
2V0_016	3.2.2.3	Completely revised
2V0_017	3.2.2.4	Completely revised
2V0_018	3.2.3.1	Completely revised
2V0_019	3.2.3.2	Completely revised
2V0_020	3.2.5.1	Completely revised
2V0_021	3.2.6	General rules added
2V0_022	3.2.6.1	Completely revised
2V0_023	3.2.6	Completely revised
2V0_024	3.3.5.3	- Table 3-14; - sample for receiving logical node address; - section below Table 3-15
2V0_025	3.3.7.2	TelIDs for MOST High Protocol removed
2V0_026	3.3.8.1	Figure 3-23; Set STX bit added
2V0_027	3.4.1.1.1	Replaced “.0.” by “.Pos.”
2V0_028	3.4.1.1.2	Completely revised
2V0_029	3.4.3.3	Equations
2V0_030	3.5.2.1	TelID and TelLen changed; One ID reserved for Ethernet frames
2V0_031	3.7.1.1	Revised (OPTypes)
2V0_032	3.7.1.2	Revised (OPTypes)
2V0_033	3.7.1.3	Revised (OPTypes)
2V0_034	3.1.4.2	Table 3-1
2V0_035	3.1.4.2.2	Revised
2V0_036	3.1.4.3.1	Handling of Isochronous data removed

Change Ref.	Section	Changes
2V0_037	3.1.4.3.6	Table 3-2; Table 3-3 added, Handling of Isochronous data removed
2V0_038	3.1.4.4.2	Completely revised
2V0_039	4.1	Figure 4-1
2V0_040	4.2.1	Completely revised
2V0_041	4.2.2	Revised
2V0_042	4.2.4	Completely revised
2V0_043	4.3	Revised
2V0_044	4.5	Completely revised
2V0_045	4.6	Completely revised
2V0_046	4.7	Completely revised
2V0_047	---	General changes in Structure: - Chapter 2.1 removed, contents included within 2.2.9 - Detailed descriptions of Control Channel (2.2) moved to 3.3 - Introduction of CMS/ AMS moved to 3.3.7 - Chapters 2.6 up to 2.12 moved to 3.2 up to 3.8 - Chapter 2.5 and 2.13 moved to Chapter 5

1 Introduction

This document is part of the specification documentation of the MOST (Media Oriented Systems Transport) system. It contains the detailed specification of the application layer, the network layer, and the MOST hardware. For an overview of the MOST system, please refer to [1].

This is the structural overview of the MOST specification documentation:



More detailed information about individual items can be found in the associated documents:

- MOST NetServices “Application Socket”
- MOST NetServices “Basic Layer”
- FOT Datasheet
- MOST Transceiver Datasheet
- MOST FunctionCatalog

This document specifies the MOST system services, which are needed to develop MOST Devices, i.e. hardware using MOST technology.

Object oriented approaches are a good means for describing complex systems in a hierarchical and clear way. Today, they therefore are the basis for most of the complex software projects. Powerful tools are available to support the development of such systems. Up to now, object oriented approaches have been used for the description of internal software structures only, but they can be used in conjunction with distributed systems as well. The description of the MOST Devices, the entire MOST System and the control flows are based on such an object oriented approach as well. For reducing the theoretical overhead, the object oriented methods have been applied only so far, as it has been useful for this application. Besides the well structured description of the system, this approach provides specification, development and documentation based on modern methods and tools for software development.

2 Application Section

2.1 Overview of Data Channels

2.1.1 Control Channel

On the control channel, data packets are transported to certain addresses, as they are on the asynchronous (packet) channel. Both channels are secured by CRC.

The control channel also has an ACK/NAK mechanism with automatic retry. It is generally specified for event-oriented transmissions at low bandwidth and short packet length. It is usable for connections with a bandwidth of approximately 10KBps, even for short periods of time.

In contrast to that, the asynchronous area is specified for transmissions requiring high bandwidth in a burst-like manner. Connections on the asynchronous channel are administered via the control channel.

2.1.2 Synchronous Channel

Continuous data streams that demand high bandwidth are transported over the synchronous channels. The connections are administered dynamically via the control channel. No bandwidth is reserved for special applications. Although synchronous connections can be built directly by source and sink nodes, it is recommended that available bandwidth be administered in a central manner, particularly in larger networks. Administration of the synchronous resources starts with the respective routines of the NetServices, via basic routines on the application level, and continues up to the administration of connections by a higher control instance.

So administration of the synchronous resources would be implemented in the ConnectionMaster function block. Since the ConnectionMaster must check to see if the connection already exists before the connection can be built, all requests for establishing connections must be directed to the ConnectionMaster.

2.1.3 Asynchronous Channel

The asynchronous channel is mainly used for transmitting data with large block size and high demand for bandwidth in a burst-like manner (graphics, some picture formats, navigation maps).

2.1.4 Managing Synch./Async. Bandwidth

On the MOST network there are 60 bytes available for synchronous and asynchronous data transfer. It is possible to divide up these resources between synchronous and asynchronous channels by means of a boundary descriptor. The boundary descriptor can be modified either by direct access to the respective register in a MOST Transceiver, or by the MOST NetServices (using MOSTSetBoundary).

The position of the boundary descriptor depends on the requirements of the system and can be changed dynamically. Supervision and changing of the bandwidth or position of the boundary is done in the Timing Master. The Timing Master is responsible for forwarding the information about the boundary's position to all nodes in the network. This task is handled automatically on the chip level. After having changed the boundary descriptor, the synchronous connections must be re-built.

2.2 Logical Device Model

The following sections describe different kinds of devices. A MOST Device contains at least one MOST Transceiver and therefore is accessible via a MOST network. MOST Device means a physical unit which can be connected to a MOST network.

2.2.1 Function Block

On the application level, a MOST Device contains multiple components, which are called function blocks, e.g., tuner, amplifier, or CD player. It is possible that there are multiple function blocks in a single MOST Device, such as a tuner and an amplifier combined in one case and connected to the MOST network via a common MOST Transceiver. In addition to the function blocks which represent applications, each MOST Device has a special function block called the NetBlock. The NetBlock provides functions related to the entire device. Between the function blocks and the MOST Transceiver, NetServices form an intermediate layer providing routines to simplify the handling of the MOST Transceiver.

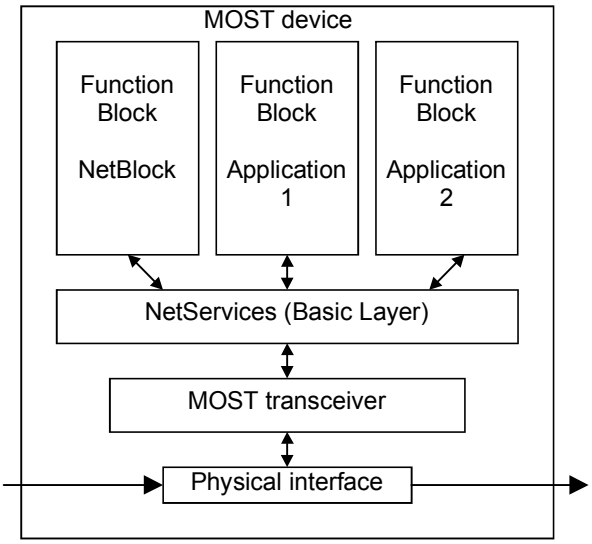


Figure 2-1: Model of a MOST Device

Each function block contains a number of single functions. For example, a CD player possesses functions such as Play, Stop, Eject, and Played. To make a function accessible from outside, the function block provides a function interface (FI), which represents the interface between the function in a function block and its usage in another function block.

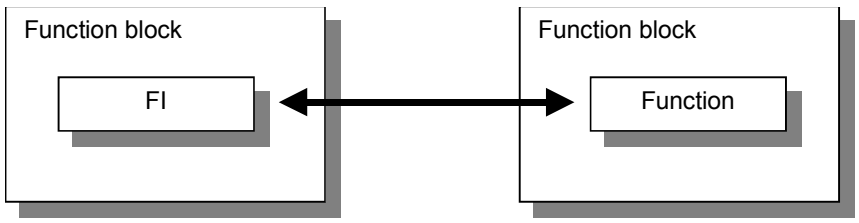


Figure 2-2: Communication with a function via its function interface (FI)

2.2.1.1 Slave, Controller, HMI

Function blocks that are always controlled are called slaves. Those function blocks that have an interface to the user are called Human Machine Interfaces (HMIs). In addition to that, there are function blocks that combine multiple functions of different function blocks to higher functions?. They control, but may also be controlled. Those function blocks are called controllers.

A clear separation between HMI, controller, or Slave cannot always be applied to devices, but in most cases a device can be classified with respect to its primary function.

2.2.1.2 First Introduction to MOST Functions

This section gives a brief introduction to the structure of MOST functions, as this knowledge is necessary to understand the following examples. Chapter 2.3 on page 32 explains the structure of MOST functions in more detail.

On the application level, a function is addressed independently of the device it is in. Functions are grouped together in function blocks with respect to their contents. Therefore, function blocks are good references for external applications to localize a certain function. A function is addressed in a function block. In order to distinguish between the different function blocks (FBlocks) and functions (Fkt) of a device, each function and function block has a name, or an identifier (ID):

FBlockID . FktID

When accessing functions, certain operations are applied to the respective property or method. The kind of operation is specified by the OPType. The parameters of the operation follow the OPType, resulting in the following structure:

FBlockID . FktID . OPType (Data)

2.2.2 Functions

A function is a defined property of a function block that can communicate with the external world, through the borders of its function block. Functions can be subdivided into two classes:

- Functions that can be started and which lead to a result after a definable period of time. This class is called “methods”.
- Functions for determining or changing the status of a device, which refer to the current properties of a device. This class is called “properties”.

In addition to that, “events” can be defined. Events result from properties, if the properties report changes by themselves (Notification).

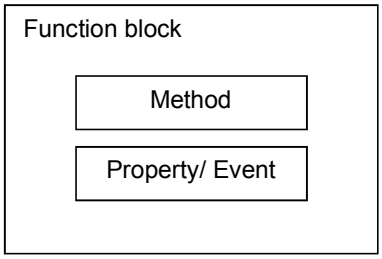


Figure 2-3: Structure of a function block consisting of functions classifiable as methods, properties and events.

2.2.3 Methods

Methods can be used to control function blocks. They are transmitted in the same way as properties. In general, a method is triggered only once, for example, starting the auto-scanning of a tuner. So method “auto-scan” is started without parameters. Of course, it is possible to use parameters, e.g., to specify the direction of auto-scan. Then only one method is needed for tune up and tune down. Especially in the case of tuners that possess automatic frequency optimization (RDS) it might be useful to specify the starting frequency for the scanning process as an additional parameter, since the currently-displayed frequency might no longer meet the frequency the receiver is tuned to. So a method’s call might contain one or more parameters.

After the reception of a method called by a function block, the respective process must be started. If this is not possible, the function block has to return the respective error message to the sender of the method call. This might happen if the addressed function block has no method of that kind, if a wrong parameter was found, or if the current status of the function block prevents the execution of the method.

After finishing the process, the controlled function block should report execution to the controlling function block (in a control device). This report might contain results of the process, for example, a frequency found by the tuner. If a process runs for a long time, it might be useful to return intermediate results before finishing, such as informing the controlling function block about the successful start of the process.

For executing methods, the following kinds of messages are exchanged via the bus:

Controller	Slave
Start of a Method with Parameters (*Start/StartResult*)	Error with cause for error (*Error*) Execution report with results (Result) Intermediate result (Processing)

The respective MOST functions needed for this messaging are described in depth in chapter 2.3 on page 32.

2.2.4 Properties

Properties can be read (e.g., temperature), written (e.g., passwords), or read and written (e.g., desired value for speed control). For each property the allowed operations are specified.

Within a function block, a property is normally represented by a variable that represents something such as a limit, or a status.

2.2.4.1 Setting a Property

The process of setting a property is described by the example of the temperature setting of a heating control.

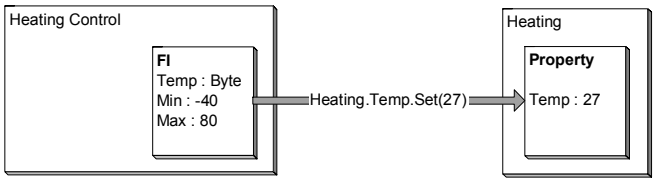


Figure 2-4: Setting a property (Temperature setting of a heating)

Function Temp is a member of the function block Heating, so the HMI sends the instruction **Heating.Temp.Set(27)** to function block Heating.

2.2.4.2 Reading a Property

In order for the HMI to display the current temperature, the value of function Temp in function block Heating must be read. Therefore the HMI sends the instruction **Heating.Temp.Get.**

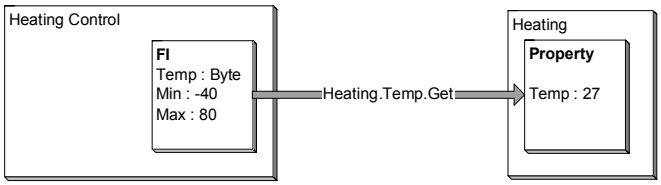


Figure 2-5: Reading a property (Temperature setting of a heating)

Heating replies by sending the status message **Heating.Temp.Status(27).**

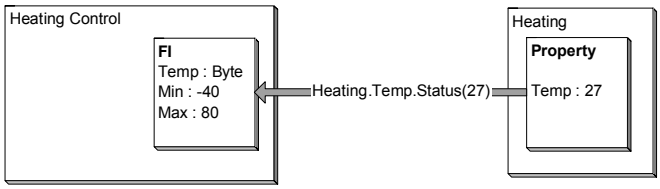


Figure 2-6: Status report of property temperature setting

For changing and reading of properties, the following types of messages are exchanged via the bus:

Controller	Slave
Setting a property (*Set/SetResult*)	Status of property (*Status*)
Reading a property (*Get*)	Error message with cause of error (*Error*)

The MOST functions needed for this messaging are described in depth in chapter 2.3 on page 32.

2.2.5 Events

Properties of a function block may change without an external influence, e.g., the temperature in the example above, or the current time of a CD player. To display current values using the functions described up to now, a cyclical reading of the properties (polling) would be required.

To reduce communication between function blocks, it would be useful if function blocks could send status reports about changes in properties without explicit requests. These are events that occur in a controlled function block, which initiate the sending of a report (notification).

Events can be used to notify reaching of limits, or the change of measured values in function blocks (e.g., the play time of a CD player has changed), or in the HMI (e.g., reception of a new value of mileage received via a CAN gateway). Events are sent only to those function blocks that have requested it by an entry in the notification matrix (refer to chapter 2.3.12 on page 93). The respective data should be transmitted in the same message with events.

2.2.6 Function Interfaces

A function interface (FI) represents the interface between a function in a function block, and its usage in another function block.

To communicate with a function, a controller or an HMI needs information about the available parameters, their limits, and the allowed operations (=FI).

In general, this information is available in the control device, and is encoded in the control program. The FI was passed on, e.g., like a device specification. To simplify the exchange of FIs, especially between different manufacturers, a formal description might be used that can be exchanged between the developers of slaves and controllers, like the well-known header files in the programming language C.

The contents of the FIs are usually known during implementation of a device (well known functions). It is also possible that FIs are transported on the bus during runtime, making it possible to dynamically reconfigure a HMI. In this way, even functions that did not exist during the development of a HMI can be made available.

Example:

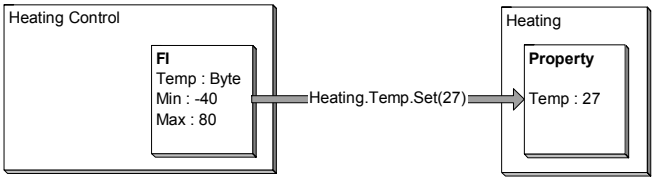


Figure 2-7: Example for a function interface (FI)

In this example the FI contains information about the data type of the function and about minimum and maximum value. In real implementations, a FI contains much more information.

During operation it is possible that a FI changes dynamically. In that case, all the function blocks that have subscribed for notification, will get the new interface description through the notification mechanism. For more information about notification, please refer to section 2.3.12 on page 93.

2.2.7 Definition Example

This section contains the example of a formal definition of a MOST Device, MyTuner, its function blocks and their methods and properties.

```
MyTuner = Device
            Tuner : TTuner;
            NetBlock : TNetBlock;
          end;
```

(Pascal Syntax)

```
Device
  {
    TTuner Tuner;
    TNetBlock NetBlock;
  } MyTuner
```

(C Syntax)

The definition specifies that MyTuner contains a function block Tuner of type TTuner, and a function block NetBlock of type TNetBlock.

TTuner is a function block and can be defined in the following way:

```
TTuner = Object
           pStation    : TStation;
           eTraffic    : TTraffic;
           pSensitivity: TSensitivity;
           mSearch     : TSearch;
         end;
```

```
Object
  {
    TStation pStation;
    TTraffic eTraffic;
    TSensitivity pSensitivity;
    TSearch mSearch;
  } TTuner
```

Here it is defined that function block TTuner contains the functions pStation (currently tuned station), pSensitivity, and mSearch (auto scan). In addition to that, the event eTraffic can be generated.

The type of function can be indicated by its name, by adding a special character to the beginning of the name (p = property, m = method, e = event).

Now property pStation will be defined as follows:

```
TStation = Property
             Frequency : Long;
             TP        : Bool;
             Quality   : Byte;
           end;
```

```
Property
  {
    long Frequency;
    Bool TP;
    Byte Quality;
  } TStation;
```

This describes pStation as a property with the parameters Frequency, TP, and Quality.

Now method mSearch will be defined:

```
TSearch = Method
            Up : Bool;
            Start : Long;
          end;
```

```
Method
  {
    Bool Up;
    Long Start;
} Tsearch;
```

Method mSearch can be started with the parameters Up for direction and Start for the start frequency.

And last, the definition of event eTraffic:

```
TTraffic = Event
             TA : Bool;
           end;
```

```
Event
  {
    Bool TA;
} TTraffic;
```

This definition specifies that event eTraffic has a Boolean parameter TA (traffic announcement).

The FI of property pStation could be defined as follows:

```
iStation = Interface
             iFrequency,
             iTP
             iQuality
           end
```

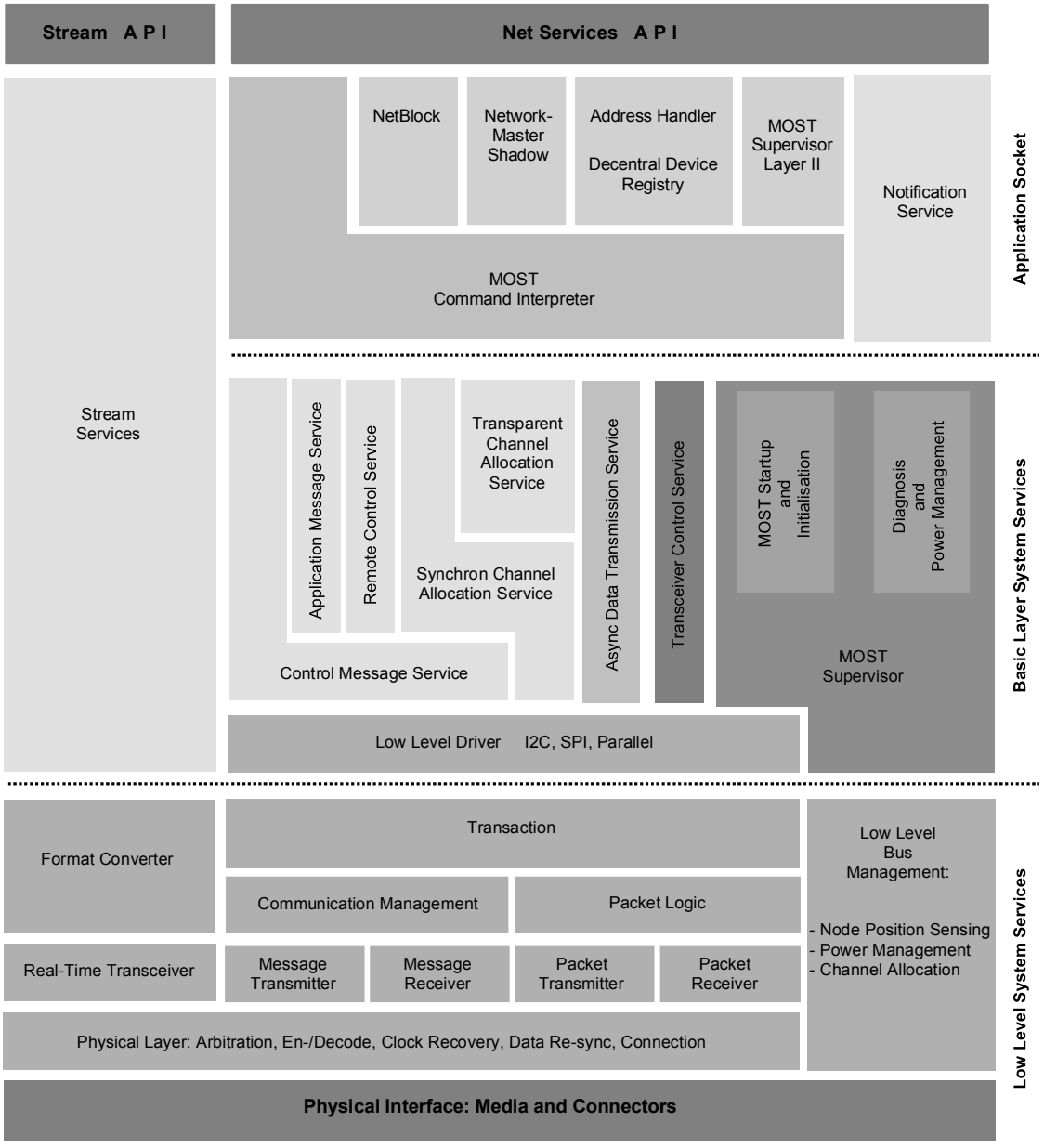
Here is the example for the interface description of parameter Frequency.

```
iFrequency = Interface
               Type : Tlong;
               Min  : 87500
               Max  : 108000
               Unit : TKHz
             end
```

The interface description of property TStation, consisting of interface definitions for the parameters Frequency, TP, and Quality, can be available as part of the device specification and can be regarded as a well known function. It can also requested by the control device and sent to it encoded in a suitable form.

2.2.8 MOST Data Flow Model

Block diagram of the MOST System Services



2.2.9 MOST System Services

The MOST System Services provide all the basic functionality to operate a MOST system. They are designed to offer the maximum flexibility with the greatest ease of use and consist of :

- Application socket
- Basic layer system services
- Low level system services
- Stream services

The low level system services, except for the physical interface, are implemented in the MOST Transceiver. Stream services, basic layer system services, and application socket are implemented in the NetServices. For more detailed information please refer to [4] and [5].

The MOST system services offer a wide variety of functions for implementing applications. Some functions or properties are mandatory for a MOST Device. MOST Devices should be able to handle control tasks in a peer to peer manner. To provide flexibility in control tasks, MOST Devices must be able to work in an environment with multiple masters.

MOST system services provide a basic framework for a MOST Device.

2.2.10 Delegation, Heredity, Device Hierarchy

2.2.10.1 Delegation

The principle of delegation provides the combining of functions of several devices to higher, distributed functions. By combining tasks and by simplifying presentation in the direction to upper layers, device hierarchies are built, which allow higher-level software to structure and control even complex system contexts in a clear way. The following example illustrates delegation.

Although car audio systems today consist of many single components, an ideal audio system would look like the one shown below, from the view of a HMI:

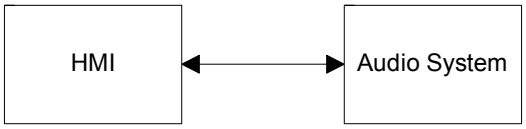


Figure 2-8: Ideal audio system

Real audio systems generally look like this:

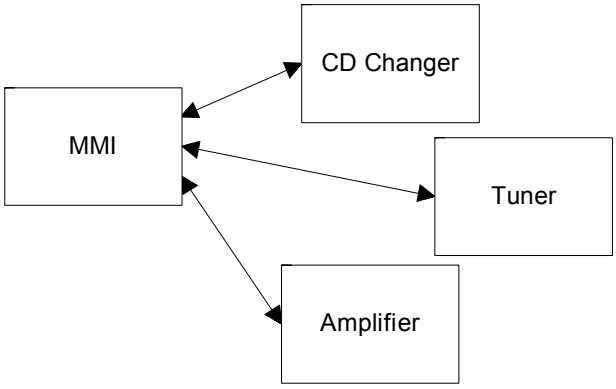


Figure 2-9: Real audio system

Coordination of the complex interaction of these components must normally be done by the HMI. This makes the design of the HMI complex and vulnerable to design changes. In addition to that, coordination of audio components requires detailed knowledge of a special range of problems. This also applies to other subsystems such as video, communication, and vehicle-based functions.

The goal of delegation is to present the audio components as one single component providing audio functions. This delegation can be related only to the direct audio area, and not to all devices having audio functionality like a navigation system or a telephone. If the audio controller were to take over complete control of these other devices, it would lead to unnecessary dependencies, making the system inflexible.

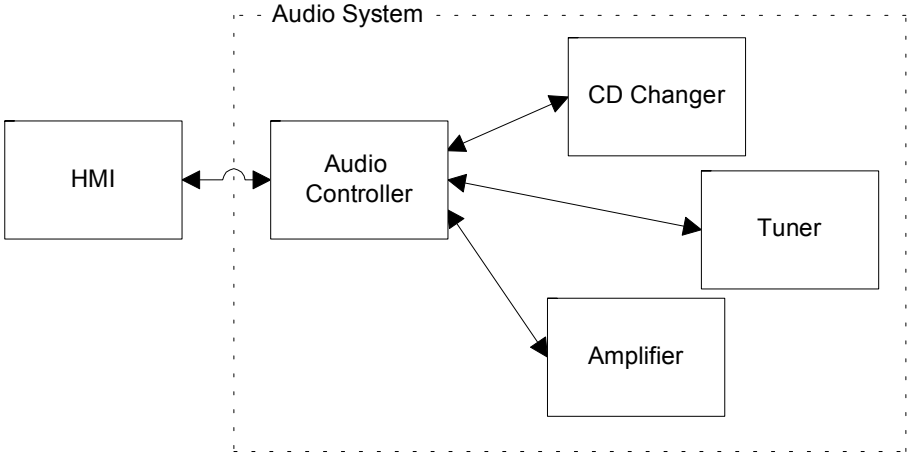


Figure 2-10: Delegation of functions of all audio components to one audio controller

By defining an audio controller, all audio functions can be provided by a single hand, even if the components are distributed. The audio controller coordinates the audio components in this case. The audio controller does not need to be a real physical device in the ring, it can be part of another device. It could even be a software module in the HMI.

The mechanism of delegation of functions is explained below by the example of a traffic announcement. The tuner device has the possibility to detect traffic announcements (TA). For a TA to be faded in during CD player plays, some steps must be taken.

Before the announcement:
Switch CD to pause, switch amplifier to tuner channels (eventually send special values for Volume and sound).

After announcement:
CD back to play, amplifier back to CD, restore Volume and sound.

The sequence and the timing of these operations must be done precisely, to avoid unpleasant effects for the listener. In order to keep the design of the HMI simple, these operations can be handled by a special unit, the audio controller. With this controller, a distributed higher TA function is built. The audio controller can provide a TP on/off, although the tuner has no such function. The interface to the HMI is represented by two simple functions TA and TP. This shows how control can be simplified by building hierarchies.

The mechanism described above creates a problem during system initialization. If the HMI looks for the function TA in the network, it would be offered by two different devices (tuner and audio controller). One possible solution to this problem is to subdivide the function blocks in hierarchy layers (ranking). This ranking must be coordinated with the entire system, to be unambiguous system-wide. A sample ranking is shown in the following table:

Class	Ranking
Slave	0
Controller	1
System Controller	2
HMI	3

Table 2-1: Ranking of device classes

To every function block, a class (with the respective ranking) is assigned during design. If a device is asked for its function blocks by a function FBlocks.Get, its answer would also contain ranking. The HMI then can identify the function block with the highest ranking as a relevant partner.

In the example above, the HMI would get the function TA from tuner (Slave) at ranking 0, and from the audio controller with ranking 1, and therefore would regard the audio controller as the relevant device.

2.2.10.2 Heredity of Functions

The example above also shows a second mechanism - the heredity of functions. The audio controller receives function TA from the tuner and hands it through to the HMI in a modified form. The TA function of the tuner is complemented by an on/off function. TA information is only passed to the HMI in case of TP = ON.

2.2.10.3 Deriving Devices/Device Hierarchy

The principle of deriving provides a system of order, where complex devices can be derived from simpler devices. The parent node provides functions and properties to all child nodes (parent is that node from which properties are inherited, while child nodes are those which inherit). In complex devices, this heredity can also be found in a respective hierarchy of classes. All devices in lower hierarchical devices (derived devices) contain all of the functions and properties of higher device classes (mandatory functions). These functions and properties can be taken from them or can be modified. Additional properties and functions can also be defined for these objects. A kind of "constructional toy" principle is generated, where complex structures are built from simpler ones.

In the figure below, the upper layer of the device hierarchy is displayed. All "intelligent" MOST Devices are derived from *MOST Device* and from *NetBlock*. This means they contain their entire functionality (e.g., a MOST chip with the properties *NodeAddr* and *LogicalAddr*) and must support all methods and properties of NetBlock and MOST Device. In addition to that, their own methods and properties are added.

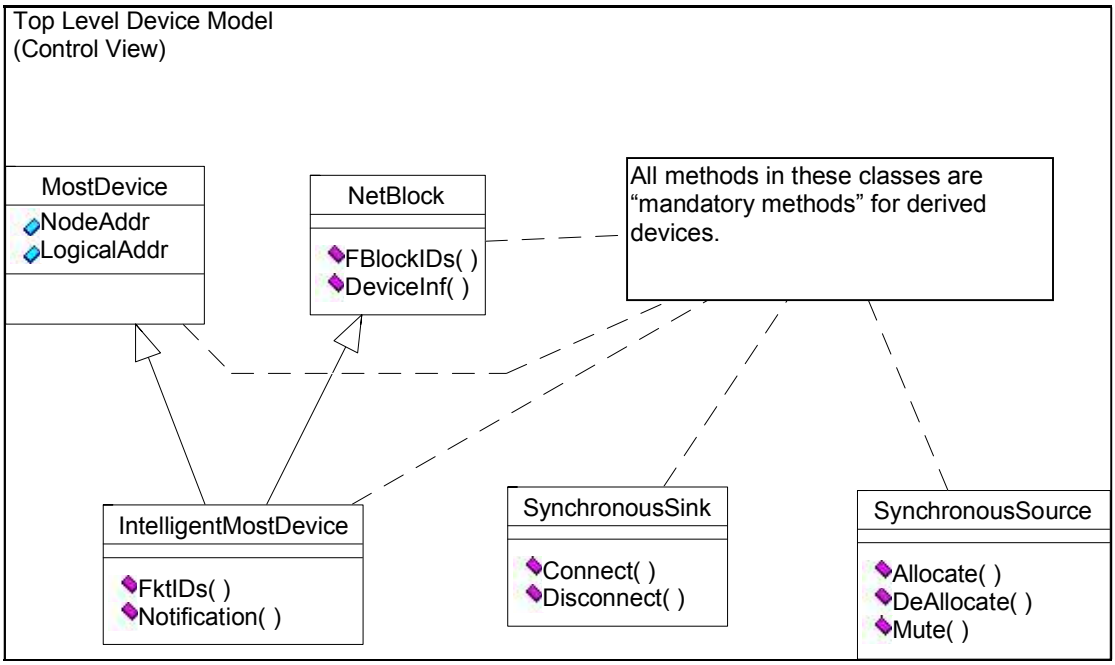


Figure 2-11: Highest layer of the device model

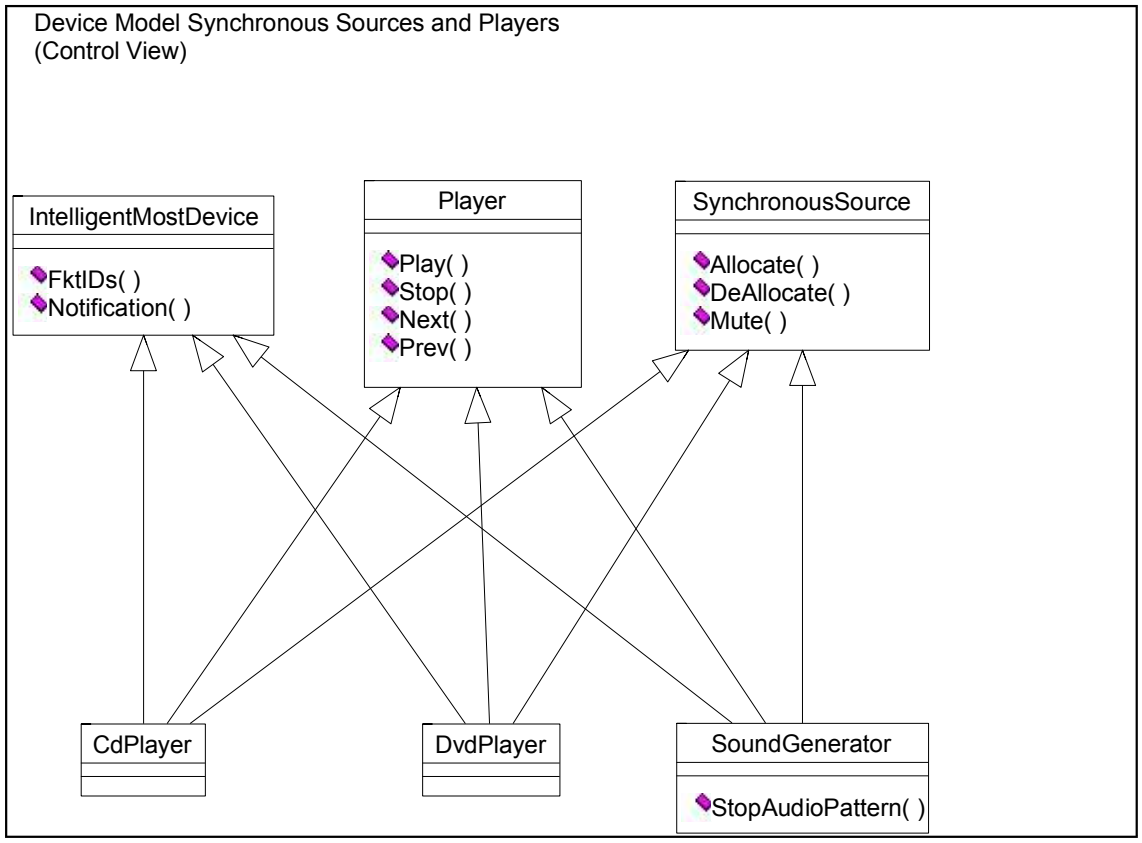


Figure 2-12: Device model for audio sources with player function

In addition to the inherited functions, the players can have their own methods and properties; for example, *StopAudioPattern* in the class *SoundGenerator*.

An application for derived devices is the designing of function or telegram catalogs for individual devices. As shown in the following table, deriving avoids re-defining all functions of the CD changer (CDMulti). They can be derived mostly from simple drives.

Device	**Group of functions**				
Master	MOST-Transceiver	+ NetBlock Master			
Slaves	%	+ NetBlock Slave			
all drives	%	%	+ Basic drive		
CDSingle	%	%	%	+ CD-Functions	
CDMulti	%	%	%	%	+ Changer

Table 2-2: Application example for the principle of derived devices

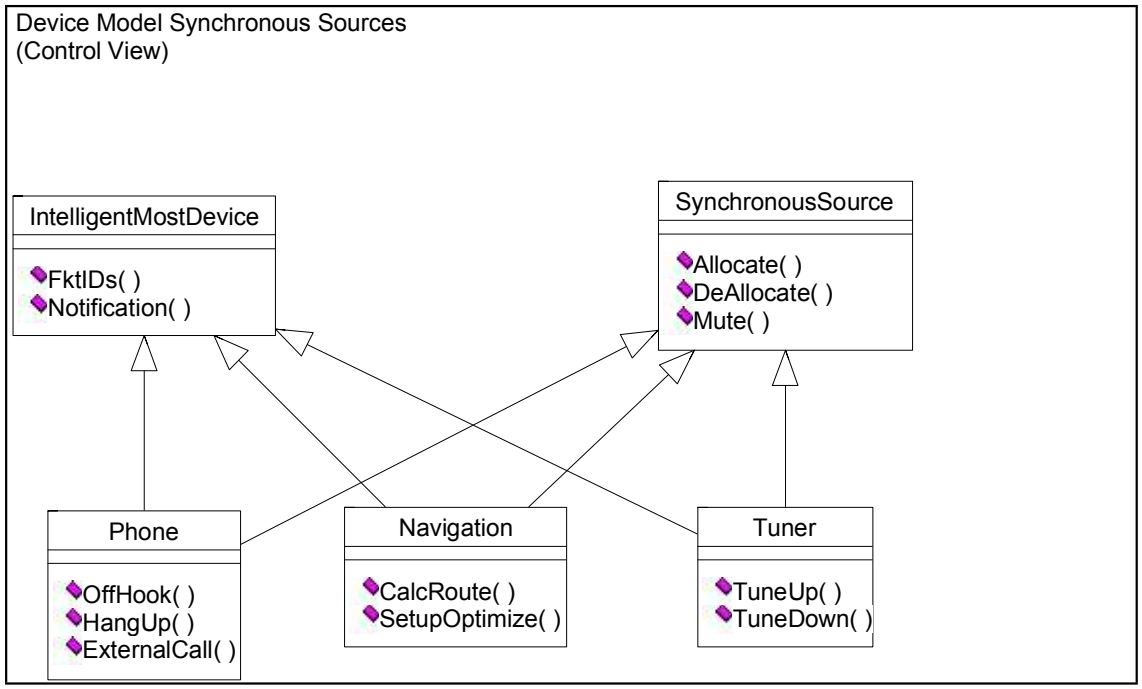


Figure 2-13: Device model for audio sources without player function

2.3 Protocols

2.3.1 Protocol Basics

As already described in section 2.2.1.2 on page 18, functions are addressed without considering the devices they belong to (on the application level). Functions are grouped together in function blocks with respect to their contents, making function blocks a good reference for localizing a certain function for an external application. A function is addressed in a function block. To distinguish between the different function blocks (FBlocks) and functions (Fkt) of a device, each function and function block has a name, or an identifier (ID) respectively:

FBlockID . FktID

When accessing functions, certain operations are applied to the respective property or method. The kind of operation is specified by the OPType, followed by the parameters of the operation. This results in this structure:

FBlockID . FktID . OPType (Data)

2.3.2 Structure of MOST Protocols

The principal structure of protocols on the application layer is:

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

In addition to section 2.2.1.2 on page 18, three components were added: InstID, Length and DeviceID. The individual elements are explained below.

2.3.2.1 DeviceID

The DeviceID stands for a physical device, or a group of devices in the network (ID is network specific and has a length of 16 bits). It precedes the protocol, and does not need to be interpreted on the application level.

If a function receives a protocol, the DeviceID contains the logical node address of the sender (DeviceID = TxAdr = TxLog). In case of an answer, it precedes the protocol as the receiver's address (DeviceID = RxAdr = RxLog). Here a group address (DeviceID = RxAdr = GroupAddress), or the broadcast address (DeviceID = RxAdr = 0x03C8) could be used too.

If the sender does not know the receiver's address, the DeviceID is set to 0xFFFF. In that case it is corrected by the Net Services.

2.3.2.2 FBlockID

The FBlockID is the name of a special function block. Every function block with a special FBlockID must contain certain specific functions. In addition to those mandatory functions, it can contain other functions. There are “System Specific” proprietary FBlockIDs, which can be used by any System Integrator (car maker). They are specific for a system and are coordinated between the OEMs developing devices for this system. A second kind of proprietary FBlockIDs is called “Supplier Specific”. Those FBlockIDs can be used by OEMs e.g. for development purpose. The special FBlockID 0xFF addresses all function blocks within a MOST device, except the NetBlock. Since this can be regarded as a broadcast function, no error status messages should be returned.

The table below shows an (incomplete) collection of FBlockIDs:

Kind	FBlockID 8 Bit	Name	Explanation
Administration	**0x0x**		
	0x00	NetServices	No FBlock, stands for telegrams that are related to network tasks, and which therefore are not passed to application layer.
	0x01	NetBlock	
	0x02	NetworkMaster	
	0x03	ConnectionMaster	
	0x04	PowerMaster	
	0x05	Vehicle	
	0x06	Diagnosis	
	0x08	Router	
Operation	**0x1x**		
	0x10	Human Machine Interface (HMI)	
	0x11	Speech Recognition	
	0x12	Speech Output Device	
	0x13	Speech Database Device	
Audio	**0x2x**		
	0x20	Audio Master	
	0x21	Audio DSP	
	0x22	Audio Amplifier	
	0x23	HeadphoneAmplifier	
	0x24	AuxiliaryInput	
	0x26	MicrophoneInput	
Drives	**0x3x**		
	0x30	Audio Tape Recorder	
	0x31	Audio Disk Player	
	0x32	ROM Disk Player	
	0x33	Multimedia Disk Player	
	0x34	Versatile Disk Player	

Table 2-3: FBlockIDs part 1.

Kind	FBlockID 8 Bit	Name	Explanation
Receiver	**0x4x**		
	0x40	AM/FM Tuner	
	0x41	TMCTuner	
	0x42	TV Tuner	
	0x43	DAB Tuner	
	0x44	Satellite Radio	
Communication	**0x5x**		
	0x50	Telephone fix	
	0x51	Phonebook	
	0x52	Navigation System	
Video	**0x6x**		
	0x60	Display	
	0x61	Camera	
	0x62	Video Tape Recorder	
Proprietary			
	0xC0..C7	**System Specific**	
	0xC8	**Reserved**	
	0xC9..0xEF	**System Specific**	
	0xF0..0xFE	**Supplier Specific**	
	0xFC	**Secondary Node**	
	0xFE	**Reserved**	
	0xFF	**All**	

Table 2-4: FBlockIDs part 2

System Specific FBlockIDs (0xC0..0xC7, and 0xC9..0xEF) can be used by any System Integrator (car maker). They are specific for a system and are coordinated between the OEMs developing devices for this system. Supplier Specific FBlockIDs (0xF0..0xFE) can be used by OEMs e.g. for development purpose.

2.3.2.3 InstID

It may happen that there are several equal function blocks (Instances) with the same FBlockID in one system (two CD changers, four active speakers, several diagnosis blocks, etc.). In order to address the function blocks unambiguously, the FBlockID is complemented by an instance ID (InstID), which differentiates equal function blocks in the system. The InstID consists of 8bits. For being able to change the InstID during operation, the NetBlock of every MOST Device has the property FBlockIDs. FBlockIDs contains the list of all FBlocks within the respective device, plus their InstIDs. The following protocol will change an InstID from value OldInstID to value NewInstID:

Controller -> Slave : NetBlock.0.FBlockIDs.Set (FBlockID, OldInstID, NewInstID)

By default, every function block has instance ID 0x00. This value is chosen, since the FBlock either knows that it is unique, or since it can not know about the existence of a “twin” in the system. In case there are several FBlocks of the same kind within one MOST Device, the default numbering (within the device) starts at 1. It is the MOST Device itself, that has to take care about the being unique of the InstIDs after Startup. The assignment of instance IDs can be encoded in a fixed way as well, or can be done e.g. by a tester, after having assembled the system.

In case there are several identical FBlocks in the system, the InstID must be assigned correctly. This means that the functional address (combination of FBlockID plus InstID) must be unique within the entire system. The NetworkMaster (application) finally is responsible for this. He takes care that the adjustment of InstIDs is finished after configuration status is ok.

In principle, as long as the InstID provides the possibility to differentiate between equal function blocks, the InstID can be chosen in any way.

Please note:
The InstID distinguishes identical function blocks. The expression “equal” means that those function blocks have the same functionality (e.g. two CD drives). This means that the basic functions are equal, but there is the possibility that they differ with respect to the total functionality (e.g. CD drive with, or without random play).

There are some special values for Instance IDs. In case of reception they will be treated by the NetServices as described below:

0x00 Don't care. The Device dispatches the message to one specific FBlock in the device. In general, this is the FBlock having the lowest InstID.

0xFF Broadcast (within a Device). The message is dispatched to all instances of the matching FBlock.

The Address Handler in the NetServices can handle only "real" InstIDs. He cannot resolve e.g. 0x00 as "don't care" for the entire system. In case a MOST Device has to send a reply, no wildcards are allowed. It always uses the correct InstID of the respective FBlock.

Please note:
InstIDs of NetBlocks are treated in a special way, since they are derived from the node position address of the MOST Device. Therefore, they start counting at 0x00.

2.3.2.4 FktID

The FktID stands for a function. This means a function unit (Object) within a device, which provides operations that can be called via the network. Examples for functions are: play of a drive, speed limit in an on-board computer, etc. On network level, the FktID is encoded in 12 bits, so 4096 different methods and properties can be encoded per function block. On the application level, the FktID is extended to 2 bytes. Exceptions to this rule will be explicitly marked.

The address range of FktIDs is subdivided in the following sections:

1. **Coordination (0x000..0x1FF)**
   Functions for administrative purposes in a function block.

2. **Mandatory (0x200..0x3FF)**
   Functions that are mandatory for the application of the function block, like the basic drive in all function blocks describing drives.

3. **Extensions (0x400..0x9FF)**
   Optional functions.

4. **Unique (0xA00..0xBFF)**
   Functions that are defined unambiguously in the entire system.
   Attention, these must be coordinated with the entire system!

5. **Proprietary/ System Specific (0xC00..0xEFF)**
   Functions, which can be used by any System Integrator (car maker). They are specific for a system and are coordinated between the OEMs developing devices for this system.

6. **Proprietary/ Supplier Specific (0xF00..0xFFE)**
   Functions, which can be used by OEMs e.g. for development purpose.

Some FktIDs in a function block that contains an application are predefined:

0x000	**FktIDs**	Reports the FktIDs of all functions contained in the FBlock (refer to section 2.3.9 on page 65).
0x001	**Notification**	Distribution list for events (refer to section 2.3.12 on page 93).

When developing proprietary Function Blocks, all possible Function IDs can be used freely, except those taken from the ranges:

Unique

Coordination

In case proprietary Function Blocks contain functions within the ranges Unique or Coordination, those functions must be in accordance to MOST Function Catalog.

Please note:
Before using any proprietary Function or proprietary Function Block, a controller must verify the identity of the device. This can be done e.g. by reading the DeviceInfo property.

2.3.2.5 OPType

This field stands for the operation which must be applied to the property or method specified in FktID:

OPType	For properties	For methods
Commands:		
0	Set	Start
1	Get	Abort
2	SetGet	StartResult
3	Increment	--
4	Decrement	--
5	GetInterface	GetInterface
6	Reserved	StartResultAck
7	--	AbortAck
8	--	StartAck
Reports:		
9	--	ErrorAck
A	--	ProcessingAck
B	--	Processing
C	Status	Result
D	--	ResultAck
E	Interface	Interface
F	Error	Error

Table 2-5: OPTypes for properties and methods

2.3.2.5.1 Error

Error is reported only to the controller that has sent the instruction. On Error, an error code is reported in the data field (Data[0]), along with additional information as shown in Table 2-6 and Table 2-7.

ErrorCode Data[0] On ErrorAck Data[2]	Description	ErrorInfo Data[1]..Data[n] On ErrorAck Data[3]..Data[n]	Description
0x01	**FBlockID not available**	--	No Info
0x02	**InstID not available**	--	No Info
0x03	**FktID not available**	--	No Info
0x04	**OPType not available**	Return OPType	Invalid OPType
0x05	**Invalid length**	--	No Info
0x06	**Parameter wrong / out of range** One or more of the parameters were wrong, i.e. not within the boundaries specified for the function. Example: Function Temp shall be set to 200, although maximum value is 80.	return Parameter	Number of Parameter (Byte containing 1,2...). Value of first incorrect parameter only (optional). Interpretation will be stopped then.
0x07	**Parameter not available** One or more of the parameters were within the boundaries specified for the function, but are not available at that time. Example: Function SourceHandles is asked for handle 0x03, which is not in use in the device at that time.	return Parameter	Number of Parameter (Byte containing 1,2...). Value of first incorrect parameter only (optional). Interpretation will be stopped then.
0x08	**Parameter missing**	--	No Info
0x09	**Too many parameters**	--	No Info
0x0A	**Secondary Node**	Return Address of Primary	Address of that node which is responsible for the secondary node sending the error
0x0B	**Device Malfunction**	--	No Info
0x0C	**Segmentation Error** After this error code, the following ErrorInfo 0x01 up to 0x07 can be sent.	0x01	First segment missing
		0x02	Target device does not provide enough buffers to handle a message of this size
		0x03	Unexpected segment number
		0x04	Target device currently has not enough buffers to handle a message of this size
		0x05	Timeout while waiting for next segment
		0x06	Device not capable to handle segmented messages
		0x07	Segmented message has not been finished before the arrival of another message sent by the same node
		0x08	Reserved, must not be used

Table 2-6: Error codes and additional information (part 1)

ErrorCode Data[0] On ErrorAck Data[2]	Description	ErrorInfo Data[1]..Data[n] On ErrorAck Data[3]..Data[n]	Description
0x20	**Function specific** After this error code, any function specific ErrorInfo can be sent. Some, with general character, are suggested here.	0x01	Buffer overflow
		0x02	List overflow
		0x03	Element overflow
		0x04	Value not available
0x40	**Busy** Function is available, but is busy	--	No Info
0x41	**Not available** Function is implemented in principle, but is not available at the moment	--	No Info
0x42	**Processing Error**	--	No Info
0x43	**Method Aborted** This error code can be used to indicate, that a method has been aborted by the Abort / AbortAck OPTypes	--	No Info

Table 2-7: Error codes and additional information (part 2)

Please note:
For avoiding infinite loops with respect to reporting errors, errors are reported only from Slave to controller. In addition, no reply of error messages is allowed on reception of broadcast messages.

In CAN systems, often define a dedicated error value (e.g., 0xFF) is for signals, to indicate the failure of the sensor, that provides the respective signal. If such a signal is read, function Sensor would report the error “Not available” (0x41). If the sensor fails, and the function has an implemented notification mechanism, the error is distributed to the registered controllers.

By OPType Error, different kinds of errors are reported. Incoming messages are scanned for all these errors one after another:

1) Syntax error:
A syntax error occurs, if e.g. a function is accessed that does not exist, or if a not implemented OPType is called. Syntax error are reported by the ErrorCodes 0x01..0x04. A syntax error will be reported directly after reception of a faulty command. This also applies to methods, which will not be started in that case. The NetServices provide automatic syntax checking and reporting.

2) Application error – Parameter error:
The specified length does not match the actual length of the data field. There have been not enough, too many parameters, or one parameter is out of range. Parameter errors are reported by the ErrorCodes 0x05, 0x06, 0x08, and 0x09. Recognition of parameter errors is the task of application layer. The NetServices give support for reporting. Messages are only accepted when being completely correct. This means especially, that the length of the parameter area must be correct. The only exception is the handling of arrays that are too short (refer to section 2.3.11.2 on page 77).

3) Application error – Temporarily not available:
In some cases it may happen, that the message is correct, but the execution is not possible at the moment. The following distinction of cases must be performed:

- It may be that both methods and properties are implemented, but cannot be executed due to operation status. An example for a method would be SMSSend of the telephone, which cannot be executed if the bus is not available. In case of being called anyhow, it would report an OPType error at error code 0x41 “not available”. In such a case, the application can supervise the status of the telephone and may repeat the sending of the SMS as soon as the network is available again.

- A method can be available, but may be busy at the moment. So it would be possible, that method SMSSend of the telephone is busy in sending another SMS. In that case an error code 0x40 “busy” would be reported. Here, the application may perform retries. This case can only occur in connection with methods.

- A property represents a memory area, which is written by Set, or read by Get. According to definition this memory area cannot be “busy”. It is solely possible that a value is within the valid range, but is not selectable at the moment. An example can be property DeckStatus of the CD drive, which cannot be set to “Play” if there is no CD loaded. This would generate an error code 0x07 “parameter not available”.

4) Application error – General execution error:
Especially when using methods, execution errors may occur. In general, such an error (unspecific; Command was correct, but execution failed) may be reported by error code 0x42 “processing error”.

5) Application error – Specific execution error:
Besides the already listed errors, a MOST application may report specific errors during execution by using OPType Error as well. Here, error code 0x20 “function specific” is used. Some possible errors are predefined for that case as well.

The examination and processing of errors is done in the logical and temporary sequence as described above and in Figure 2-14 (below).

6) Application error – Error secondary node:
Detailed information about Secondary Nodes is to be found in section 3.9 on page 181. In case a secondary node receives any control message, it replies with an Error „secondary node“ (ErrorCode=0x0A). The reply contains the address of the primary node (=ErrorInfo), which is responsible for that secondary node. In addition to that, the reply shall appear to be sent by NetBlock (with FBlockID = 0x01, InstID = Pos) and from NetBlock's function FBlockIDs (FktID = 0x000).

Target Address	FBlockID	InstID	FktID	OpType	Length	Data[0]	Data[1]	Data[2]
Received Source Address	01	Pos	000	F	03	0A	Node Addr. (HB) of Primary Node	Node Addr. (LB) of Primary Node

Table 2-8: Structure of an error message containing an “Error Secondary Node”

In general, a Secondary Node reacts only upon messages that have a receive type less than 0x02. The InstID here, is derived from the NetBlock’s InstID, which is equal to the position of the node in the ring (Pos).

6) Application error – Device Malfunction:
This error indicates device malfunction and provides to distinguish between a generally broken device, or a temporarily being unavailable of a Device.

7) Application error – Segmentation error:
A MOST System provides the option of transporting messages that exceed the length limitations given by the control channel of the MOST bus (17 Bytes). This is done by dividing the message up into several segments. Each of the segments then is transported as one control channel telegram to the receiver. In order to make sure that the data can be reassembled safely on the receiver's side, each telegram carries the appropriate additional information in its protocol header (TelID, Segment counter).

Errors during reassembling the original message in the receiver can be caused e.g. by missing segments, wrong order of arrival or exceeding the timeout between two segments. In case of such an error, the parts of the message that have already been received are discarded. In addition, the application layer within the receiver is notified by the NetServices through a callback function.

The segmentation error notifies the sender about the failure of the segmented transfer. Therefore, the sender’s application may react in appropriate way, e.g. by retrying to send the same message again. The reaction depends on the respective problem that caused the error.

This „Segmentation Error“ shall be sent back to a controller that failed to send a segmented message to a Slave. Error messages can only be directed from the Slave to the controller, to avoid infinite loops of error messages. So a failure in sending a segmented message from the Slave to the controller will not be notified to the Slave. In this case, it is the responsibility of the controller application to take appropriate measures, as soon as it is notified about the error by the callback function mentioned above.

8) Application error – Method Aborted:
This error is used in case of abortion of methods by OPType abort or AbortAck. A MOST Device called "A" starts a method in Device "B" by using StartResult. Due to some exceptional events, a third Device "C" aborts that method running in Device B, which was started by Device "A". In that case, Device B reports error "Method Aborted", to Device A. Please note, that methods in general should be aborted only by that application, which has started the method.

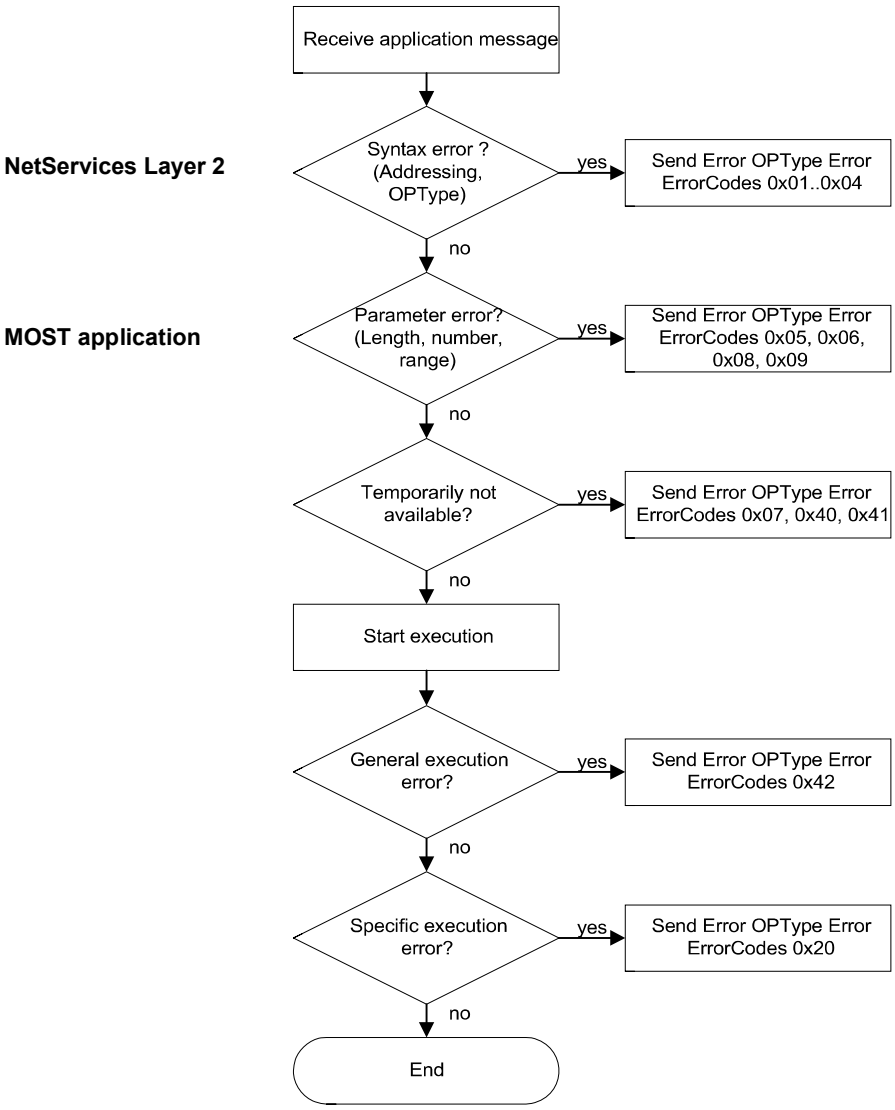


Figure 2-14: Processing of messages including error check on different layers.

Of course there can exist errors on application level that do not appear in the MOST syntax (i.e. reported by OPType Error). An example would be the processing of errors within a data transfer in TCP/IP. From the point of view of the MOST system, such a data transport is only the transport of data packets to a receiving function. The contents of the packets and the fact whether that data contains errors is interpreted on application level only. Higher levels of error management and individual error messages are to be specified individually.

Please note:
The error messages described here, mainly serve the purpose of debugging. They should be handled in a controller only, if the system's performance requires it. Otherwise error processing should be omitted, and the devices should be designed as failure tolerant systems. With respect to that, the slaves also should manage with the existing error messages. Individual error messages using error code 0x20 should be avoided if possible.

2.3.2.5.2 Start, Result, Processing, Error

By using Start, a controller triggers a method. In opposite to StartResult, it “trusts” in the execution and therefore does not expect a reply. So neither a reply using Processing, nor one using Result will be generated. This approach is useful only for Methods that do not return results.

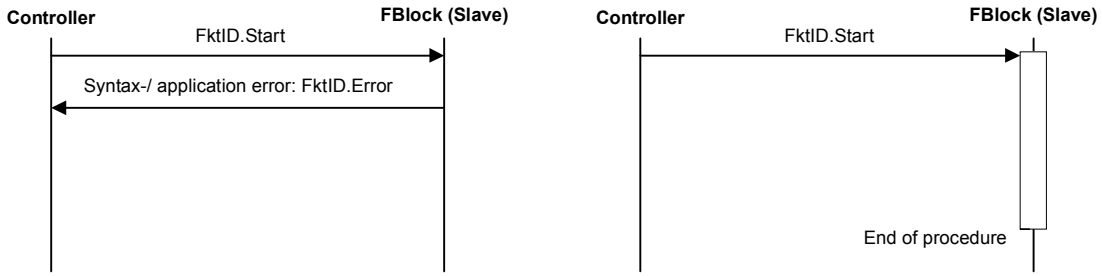


Figure 2-15: Sequences when using Start with and without error

Please Note:
A method started by "Start" must be called only one time (no multiple instances are allowed). In case a method, that was started by "Start", is currently running, and a second controller tries to start the same method again, the method has to reply an error "Busy". For running several instances of the same method, StartAck and ResultAck must be used.

2.3.2.5.3 StartResult, Result, Processing, Error

In opposite of triggering a method by using Start, the controller requires feedback when it uses StartResult. It then expects reports about the currently running procedure (with Processing), as well as about the Result (Result or error). If a method does not return a result by parameters, it returns Result() as a signal of a successful processing.

If there are syntax or parameter errors during the calling of a method, there will be a reply using Error. The method will not be started.

If a method that was started can generate a result within 100ms after reception of StartResult, it returns the result by using “Result(<Parameter>)” as soon as it is available. There will be no reply “Processing” in that case. The same applies to application errors.

If a method can not generate a result 100ms after having received StartResult and if there is no application error, it replies after those 100ms by using “Processing”. After that the timer will be re-started and the procedure starts again. That means that in case of terminating the method within the 100ms, a reply “Result(<Parameter>)” will be sent. Otherwise “Processing” will be reported.

The controller evaluates the replies by using a timer interval of 200ms (compensation of eventual delays). In case that there is no reply within these 200 ms (Neither Result, nor Error, nor Processing), it assumes an error.

It is allowed to System Integrators, to increase the timeout value (100ms) for acknowledging the start of a method. This is to be done individually for the respective function, within the function catalog.

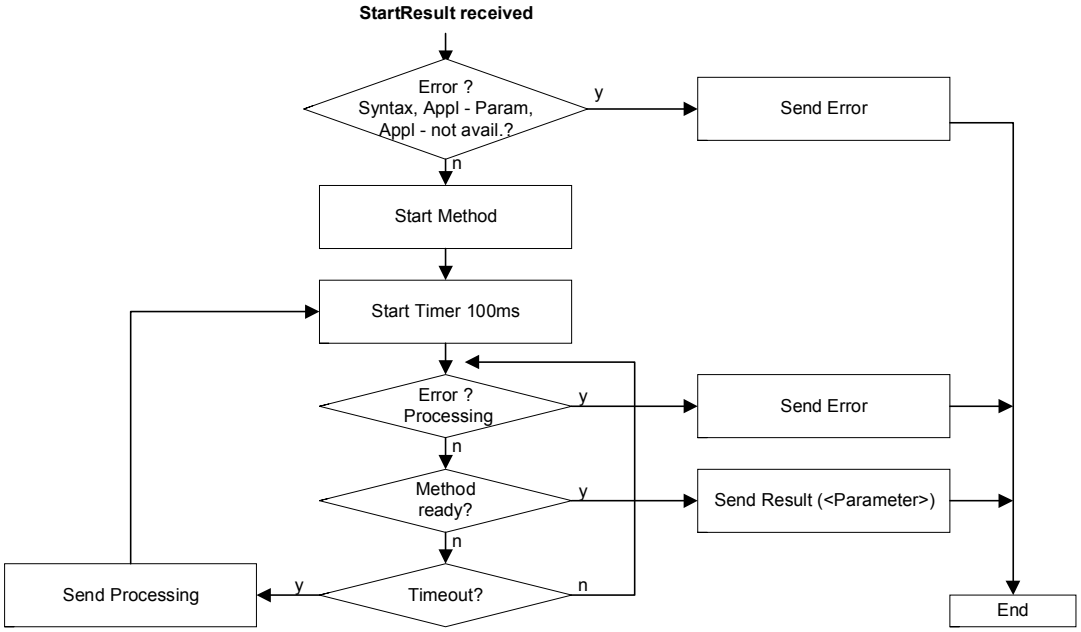


Figure 2-16: Flow for handling communication of methods (slave's side)

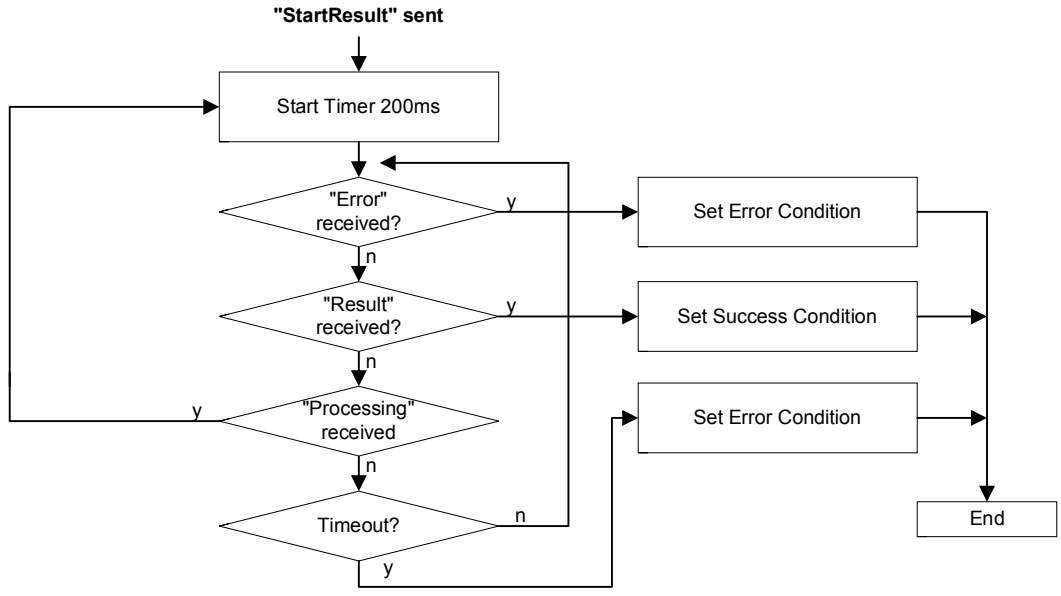


Figure 2-17: Flow for handling communication of methods (controller's side)

2.3.2.5.4 StartAck, StartResultAck, ProcessingAck, ResultAck, and ErrorAck

The behavior is equal to that of Start, StartResult, Processing, Result and Error (refer to section 2.3.2.5.3 on page 43). The only difference is, that the first parameter transports the SenderHandle (refer to section 2.3.6.2 on page 61).

2.3.2.5.5 Get, Status, Error

By using OPType Get, a controller asks for the status of a property. In case of a request by using Get, a reply using Status will be generated, if the syntax check has shown no errors. Otherwise Error will be returned. If the controller does not receive any reply 200ms after having sent Get, an error can be assumed. It is not critical, if the controller reacts more tolerant and waits for a longer time. Nevertheless, an interruption of the waiting process is a must.

2.3.2.5.6 Set, Status, Error

By using Set, the contents of a Property is changed. Set behaves equal to Start. This means that the controller “trusts” in the execution of its command and does not expect any reply (except eventual error reports). If the syntax check is ok, the command can be executed.

The changed status of the property will be reported to all controllers that are registered for this function. This is done via Notification. If the triggering Controller is registered, it will receive a status report indirectly. This way is recommended, e.g. if the controller is registered anyhow in the Notification Matrix. In addition to that it may be, that the changing of a property by a controller from outside, generates the changing of the status of several other properties by some internal mechanisms.

Therefore the controlling of properties by using Set is the preferred mechanisms for Controllers, that are registered in the Notification Matrix of a controlled function block.

2.3.2.5.7 SetGet, Status, Error

SetGet is the preferred way of controlling function blocks, for Controllers:

- that control a property only in rare cases
- which are not registered in the Notification Matrix

SetGet is a combination of Set and Get, which means that the Controller (in case of a correct syntax) automatically gets the changed status in return. This is independent of the Notification Matrix.

2.3.2.5.8 GetInterface, Interface, Error

These OPTypes can be compared with Get, Status and Error (refer to section 2.3.2.5.5). Instead of the status, the Function Interface will be requested.

2.3.2.5.9 Increment And Decrement

Increment and Decrement provide a relative changing of a variable in opposite to the absolute changing by using Set. When using Increment or Decrement, the new status will be reported to the triggering Controller as well as to the Controllers registered in the Notification Matrix. This is similar to SetGet. In case of a Controller requesting Increment or Decrement although the respective maximum or minimum is reached, no error will be reported. In fact the (old) new value will be reported. This answer is directed to the triggering controller only. A reporting to the controllers registered in the Notification Matrix is not required, since the value actually did not change.

2.3.2.5.10 Abort

This OPType is available for methods only. When used, Abort interrupts the execution of a method. It behaves like Start, which means that the Controller “trusts” in the execution and does not expect a reply. Please note, that methods in general should be aborted only by that application, which has started the method.

2.3.2.5.11 AbortAck

This OPType is available for methods only. When used, AbortAck interrupts the execution of a method. It behaves like Start, which means that the Controller “trusts” in the execution and does not expect a reply. In opposite to “Abort”, AbortAck transports additional “routing” information (SenderHandle, as described in section 2.3.6.2 on page 61). Please note, that methods in general should be aborted only by that application, which has started the method.

2.3.2.6 Length

Length specifies the length of the data field in bytes. It is encoded in 16 Bits.

Length = 0x00 00 Data field of length 0
Length = 0x00 01 Data field of length 1 byte ...
Length = 0xFF FF Data field of length 65535 Byte ...

Functions that need to transport voluminous application protocols communicate via MOST High Protocol and the packet data transfer service. These functions will be marked in the function catalog.

Please note:
Length is not transmitted directly via MOST, but is reconstructed from the number of received telegrams and the TelLen at the receiver's side.

2.3.2.7 Data And Basic Data Types

In principle, the data field of a protocol on the application layer might have any length up to 65535 bytes. In a telegram on the control channel of the MOST bus, the maximum length is 12 bytes. So longer protocols must be segmented, i.e., be sent divided up in several telegrams. It should be kept in mind that even on the application level, the data fields of a protocol should exceed 12 bytes only in exceptional cases.

Within a data field, none, one, or multiple parameters in any combination of the following data types can be transported. They are transported MSB first. The sign is encoded in the most significant bit and 2's complement coding is used for signed values. There are the following basic data types:

- Boolean
- BitField
- unsigned Byte
- signed Byte
- unsigned Word
- signed Word
- unsigned Long
- signed Long
- Enum
- String
- Stream

Since MOST provides enough bandwidth, parameters are transmitted in a way that can be displayed directly. Using only the data types mentioned above, no floating point format would be possible. The missing information about the location of the decimal point is added via an exponent of type signed byte. The value to be displayed must be transported in the following way:

value to be displayed = transmitted value * $10^{Exponent}$

Example 1:

transmitted value :	1073 (word)
exponent:	-1
step:	1
unit:	MHz
value to be displayed :	107.3 (MHz) (can be changed in steps of 100kHz)

In case of an Increment operation with NSteps = 5, the current frequency would be incremented from 107.3MHz to 107.8MHz.

Example 2:

transmitted value :	1073 (word)
exponent:	+5
step:	1
unit:	Hz
value to be displayed :	107,300,000 (Hz) (can be changed in steps of 1Hz)

In case of an Increment operation with NSteps = 5, the current frequency would be incremented from 107,300,000Hz to 107,300,005Hz.

Example 3:

transmitted value :	1000 (word)
exponent:	-3
step:	10
unit:	m
value to be displayed :	1.000 (m) (can be changed in steps of 10mm)

In case of an Increment operation with NSteps = 5, the current length would be incremented from 1.000m to 1.050m.

The exponent can be already known to the receiver of the parameter (controller), or it can be requested by the sender (function) of the value (refer to section 2.3.11 on page 67). It is not transported together with the parameter.

2.3.2.7.1 Boolean

Definition of Type	Comments
1 Byte	Only one bit can be used in each byte.

2.3.2.7.2 BitField

Definition of Type	Comments
Size Byte	= (Mask.Data)

Size: - 1, 2, 3...n

Data: ½ Size Byte (Data Content Area)

Mask: ½ Size Byte (Masking Area):
"Mask" is a masking bit field of the same size as the Data Content Area "Data". It indicates to which bits in the Data Content Area of the BitField an operation shall be applied. The LSB of "Mask" masks the LSB of the Data Content:

Bit k (Mask) = 1 -> apply Operation to Bit k (Data)
Bit k (Mask) = 0 -> do not apply operation to Bit k (Data)

Example:

```
State:      MyBitField.Status (XXXX XXXX, 1010 1001)
Operation:  MyBitField.Set    (0000 1000, 1010 0111)
NewState:   MyBitField.Status (XXXX XXXX, 1010 0001)
```

X = don't care

2.3.2.7.3 Enum

Definition of Type	Comments
1 Byte	-

2.3.2.7.4 Unsigned Byte

Definition of Type	Comments
1 Byte	-

2.3.2.7.5 Signed Byte

Definition of Type	Comments
1 Byte	-

2.3.2.7.6 Unsigned Word

Definition of Type	Comments
2 Byte	-

2.3.2.7.7 Signed Word

Definition of Type	Comments
2 Byte	-

2.3.2.7.8 Unsigned Long

Definition of Type	Comments
4 Byte	-

2.3.2.7.9 Signed Long

Definition of Type	Comments
4 Byte	-

2.3.2.7.10 String

Definition of Type	Comments
Variable length	= (Identifier.Content.Terminator)

Please note:
In general, only "MSB first, High Byte first" notation must be used for strings. Every string starts with an Identifier and is Null terminated.

Identifier:	1 Byte	0x00	: UNICODE 16 bit
		0x01	: ISO 8859/15 8 bit
		0x02	: UTF8
		other values	: ASCII
Content:	-	Characters	
Terminator:	1 Character	Null character. Number of zeros. Depends on encoding.	

For calculating length, only the number of characters is relevant. Length explicitly excludes the Identifier and the terminating character(s). The encoding of an "empty" string depends on the used code:

Code	"Empty" String	Comment
UNICODE 16 bit	0x00,0x00,0x00	-
ISO 8859/15 8 bit	0x01,0x00	-
UTF8	0x02,0x00	-
ASCII	0xkk,0x00	0xkk is any "other" value, as defined above.

2.3.2.7.11 Stream

Definition of Type	Comments
Any Data	-

2.3.3 Function Formats in Documentation

The protocol structure that was described above is valid for communication within devices, between the NetServices and a function block containing an application. The protocols have different DeviceIDs, depending on the protocol being received or transmitted. In documentation that must be human readable, the following general description must be used, which covers both cases:

```
SrcAdr -> TrgAdr: FBlockID.InstID.FktID.OPType.Length(Parameter)
```

SrcAdr and TrgAdr are the physical MOST addresses of the sending and the receiving device, respectively. On the sender's side, it is identical with the TrgAdr, and on the receiver's side with the SrcAdr (please refer to the example in section 2.3.5 on page 53). In most cases, only one instance of the function block is available in the system, and InstID can be omitted. Descriptions can also be simplified by omitting the Length.

```
SrcAdr -> TrgAdr : FBlockID.FktID.OPType(Parameter)
```

Example:

Choosing track of the CD changer:

```
HMI -> CDC : AudioDiskPlayer.Track.Set(5)
CDC -> HMI : AudioDiskPlayer.Track.Status(5)
```

2.3.4 Protocol Catalogs

The telegrams are included in a catalog and are grouped by functions (Function catalog). It is a good approach to implement this catalog in a database, so that a printable version can be produced.

The tools of Oasis SiliconSystems (e.g., OptoLyzer*4*MOST® Professional, MOST RapidControl) use a special form of syntax tree for automatic generation and disassembling of messages. In order to generate syntax trees automatically, these tools have a link to the catalog database.

2.3.5 Application Functions on MOST Network (Introduction)

The controlling mechanisms described in this document are generally independent of the kind of bus used. Protocols on the application level are described in a universal way. They are transported virtually from one application to the other. In reality they are transmitted with the help of a bus system, here the MOST network, which is described in detail below.

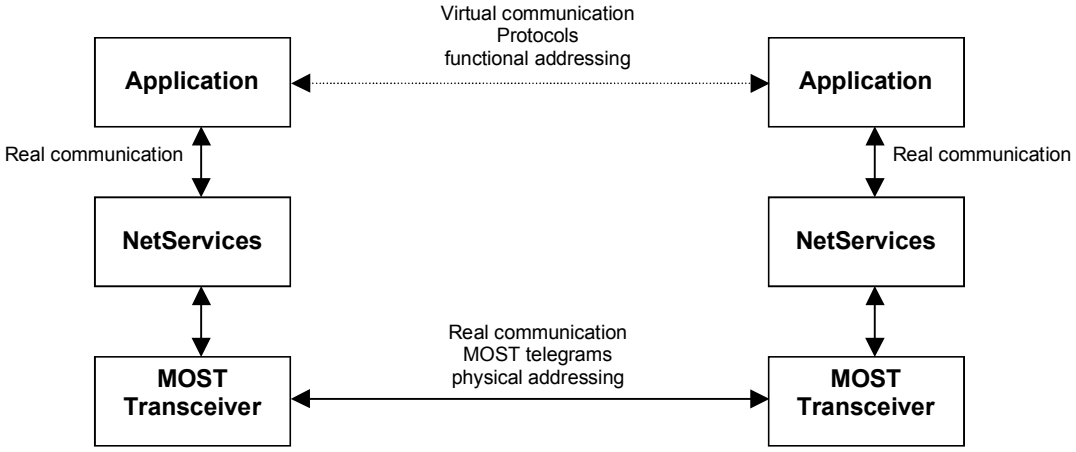


Figure 2-18: Virtual communication between two devices on application layer and real comm. via network

All application protocols are finally transferred via the control channel of the MOST network. From the application's point of view, all protocols are passed on to the NetServices. Depending on the length, an application protocol is sent with a single transfer if it fits into one MOST telegram, otherwise via segmented transfer.

In a MOST network, nodes, or devices, are addressed. In order to transport a protocol to a function block, the MOST telegrams are provided with the address of the device that contains the function block.

Here, the entire data flow of an interaction between two devices via the network layer is described. One device controls the functions of the other. The figure below shows the properties of a function block with the FBlockID CD and the InstID 0 (Any instance). The function block is found in device CD Player with the physical MOST address CDC.

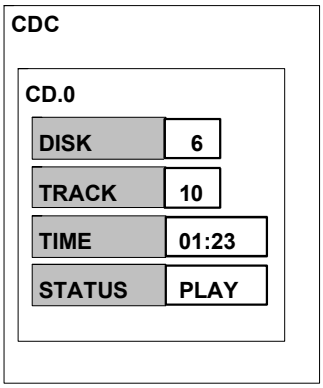


Figure 2-19: Device with MOST address CDC, a function block CD Player with FBlockID CD, and its functions.

For example, another track can be chosen by reception of the following protocol:

```
CD.0.Track.Set(10)
```

This protocol is sent by a device with the physical MOST address HMI. Therefore it will be passed on to the NetServices in the following form:

```
FFFF.CD.0.Track.Set(10)
```

The first part is a special DeviceID, which means that the physical address of the receiver is not known on the application level. The NetServices will complement the address. The result is:

```
CDC.CD.0.Track.Set(10)
```

For transmission this is complemented by the sender's physical address:

```
HMI.CDC.CD.0.Track.Set(10)
```

Since the receiving device knows its own physical address, this address does not need to be passed on to the application level. The received protocol therefore looks like:

```
HMI.CD.0.Track.Set(10)
```

If the function wants to report its new status, it builds the following protocol:

```
HMI.CD.0.Track.Status(10)
```

Based on this, the NetServices builds the following telegram:

```
CDC.HMI.CD.0.Track.Status(10)
```

In the HMI the receiver's address is removed and the protocol is passed to the application:

```
CDC.CD.0.Track.Status(10)
```

The general data flow via the different layers in the two devices is displayed in the following figure:

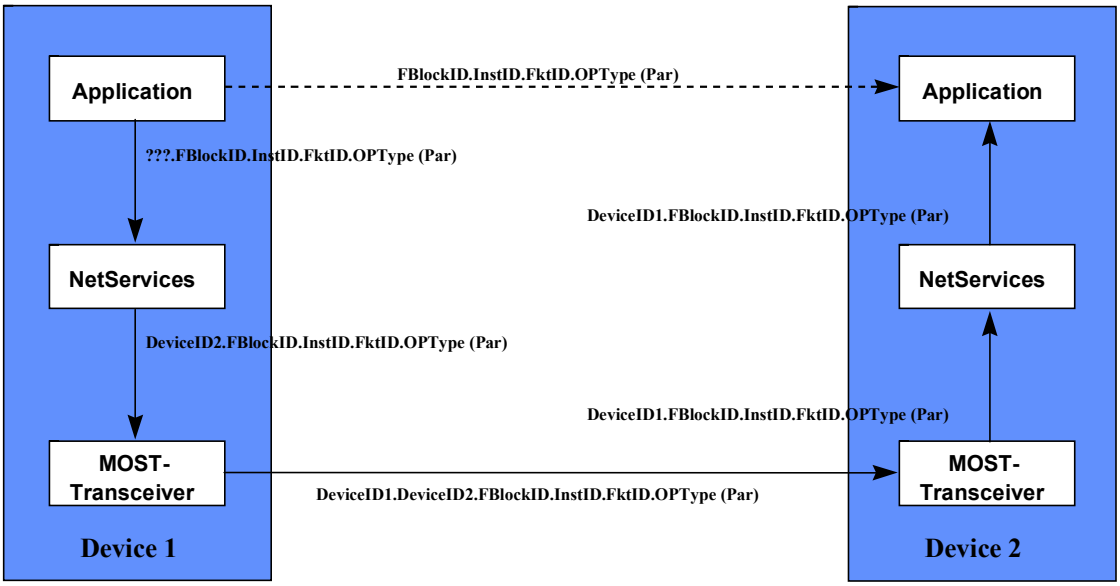


Figure 2-20: Communication between two devices via the different layers

2.3.6 Controller/Slave Communication

For communication between Controllers and Slaves, properties and methods must be differentiated.

2.3.6.1 Communication With Properties Using Shadows

Below, communication between a controlling and a controlled device is explained for Properties by an example:

- Controlling device (Controller):
  Contains a function block controlling another function block.
- Controlled device (Slave):
  Contains only controlled function blocks (for demonstration purpose).

The properties of a device should describe the current operation status completely at any time. The figure below shows the properties of a function block CD changer with FBlockID CD and the unspecific InstID 0 in the device with the MOST address CDC.

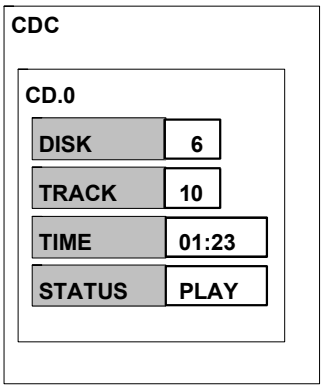


Figure 2-21: Example for a Slave device

Operation status of the player is determined by the properties Disk (number of loaded CD), Track, Time, and Status (Play, Stop, Forward, Rewind, Eject). By changing these properties the player can be controlled by another device.

For example, another track can be chosen by sending the following protocol:

```
HMI->CDC: CD.0.Track.Set(10)
```

If this operation is successful, the new state of the CD player is confirmed by the following protocol:

```
CDC->HMI: CD.0.Track.Status(10)
```

By sending this protocol, the player can be stopped:

```
HMI->CDC: CD.0.Status.Set(Stop)
```

Also in this case, the new state of property Status can be transmitted via a protocol:

```
CDC->HMI: CD.0.Status.Status(Stop)
```

These status messages are sent by the CD player even in a case where a property changes itself, e.g., when the player changes to the next track during play mode (on the condition that another device is registered in the notification matrix of function block CD).

The MOST device address of the CD changer (represented by the abbreviation CDC) together with FBlockID and the InstID describe the property to be changed. To make sure that the protocols for controlling a device find their way through the system, the property description must be unique in the entire system.

If there are multiple CD players in the system, they get different InstIDs, and in addition to that, different MOST addresses. Based on that, two players can be controlled by a HMI in the following way:

```
???->CDC1: CD.1.STATUS.SET(STOP)
???->CDC2: CD.2.STATUS.SET(STOP)
```

By this, two CD function blocks can be addressed unambiguously, even if they are located within one physical device with one MOST address. This also guarantees that status reports can be assigned unambiguously:

```
CDC ->???: CD.0.STATUS.STATUS(STOP)      Status of CD in CDC
CDC1->???: CD.1.STATUS.STATUS(STOP)      Status of CD in CDC1
CDC2->???: CD.2.STATUS.STATUS(STOP)      Status of CD in CDC2
CDC->???: CD.1.STATUS.STATUS(STOP)       Status of 1st Player in CDC
CDC->???: CD.2.STATUS.STATUS(STOP)       Status of 2nd Player in CDC
```

The controlling device (controller) contains the Shadows of the functions it controls. The Shadow of a function in the control device represents an image of the property of the Slave device. That means, for each controlled property of the Slave device, the control device contains a respective variable. For the controller, the function seems to reside in its own memory area. This is shown in the figure below:

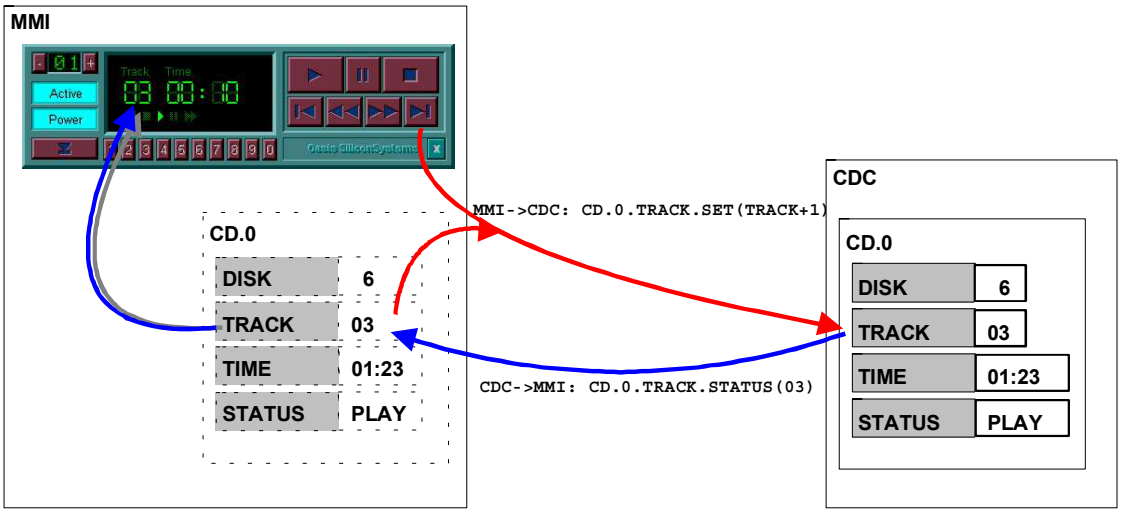


Figure 2-22: Virtual illustration of the controlled properties in the control device

The HMI shown in Figure 2-22 has an image of all properties of CDC (Slave device) represented by the variables Disc, Track, Time and Status. These variables are required to store the display values, and can be used for control purposes too. The example shows the flow of communication when using the “Next track” button.

On a click onto the button, the HMI takes the contents of its local variable Track, increments it by one, and sends the protocol CD.0.Track.Set(Track+1) to device CDC. After the player has changed track, it replies by sending protocol CD.0.Track.Status(3). Addressing of the response is equal to the addressing of the command, except the address, since the answer is sent to a (virtual) identical function block. Variable Track reacts only on that protocol and stores the new value. The change of variable Track causes the HMI to update its display.

As shown in the figure below, there is one protocol assigned to each variable unambiguously. Every variable in HMI “reacts” only on the assigned protocol, sent from the respective device.

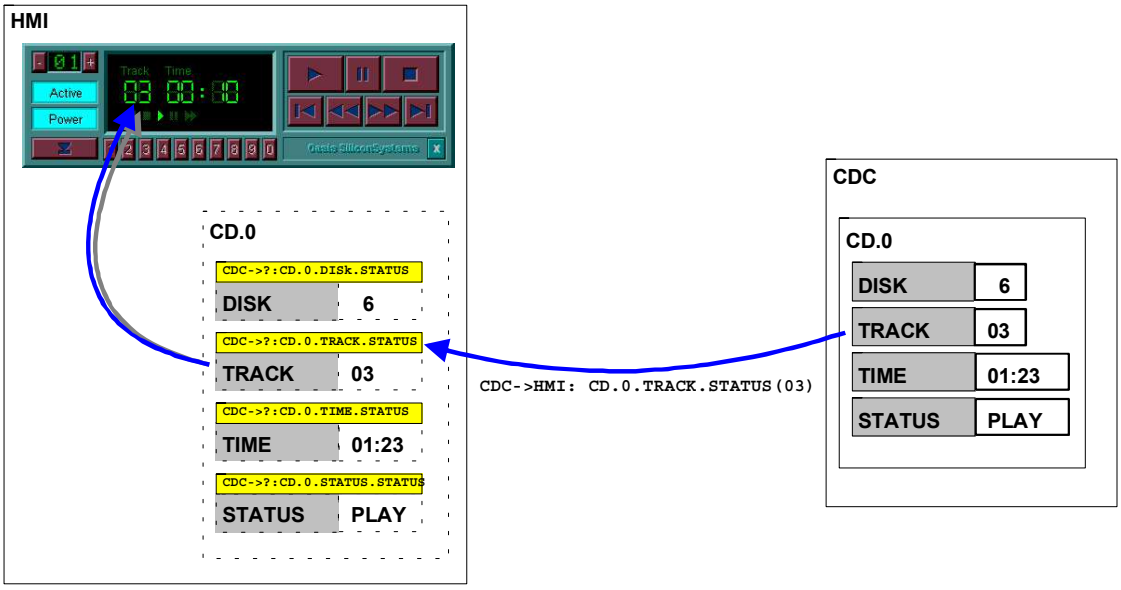


Figure 2-23: Unambiguous assignment between protocol and variable.

The figure below shows the advantage of this approach when controlling multiple devices. The HMI has an image of the controlled CD player, as well as an image of the tuner. Even during play operation of the CD player, the tuner sends status changes to the HMI. In CD operation mode, this information is not shown on the display, but is stored in the respective variables. This means that the current information about the tuner is available immediately if the operation mode is changed from CD to Tuner, with no extra polling needed.

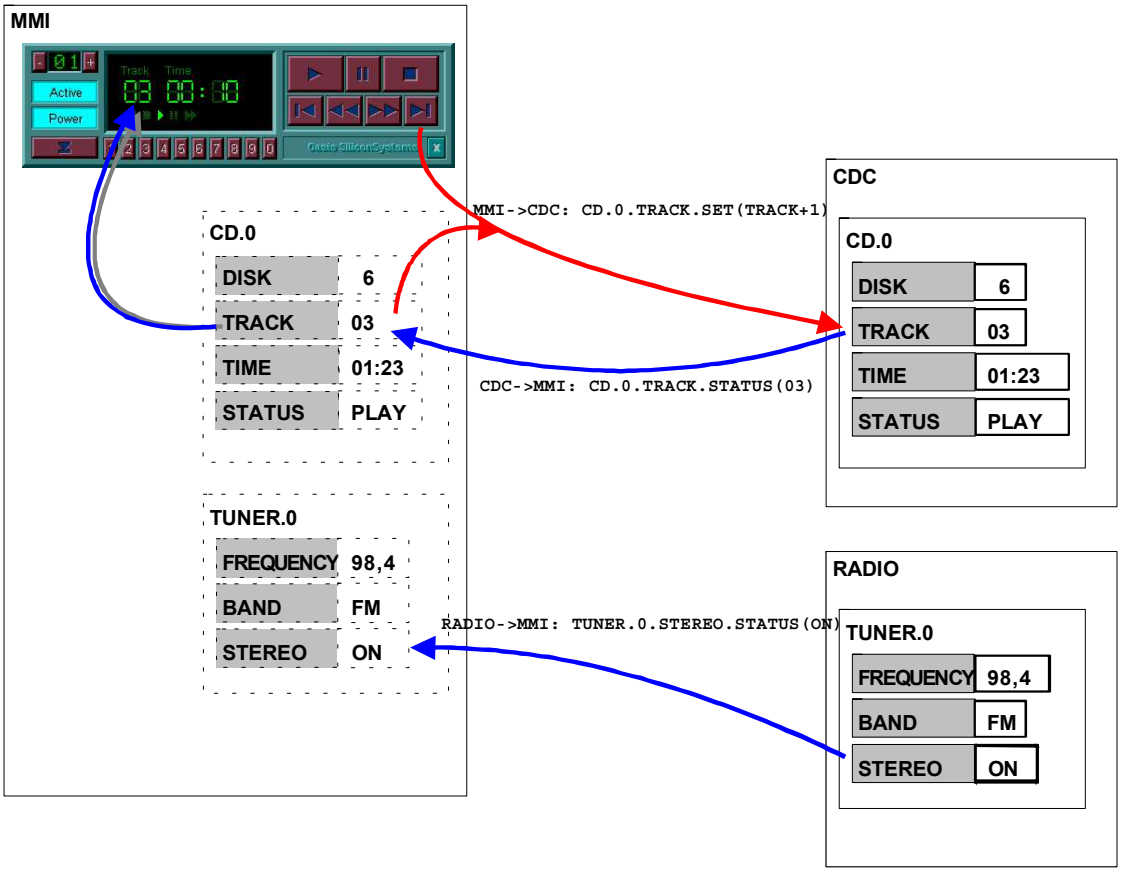


Figure 2-24: Controlling multiple devices.

A similar case could be imagined, if several identical CD players are available in the network. Operation mode of the HMI could be changeable, for example, between CD1 and CD2. The display would show only the status of the currently selected player, and the keyboard would be switched, too.

As shown in the graphic below, such a HMI would contain two sets of variables (shadows), one for each CD player. The variables for CD.1 react only upon protocols of CD.1, while the variables for CD.2 react only upon protocols of CD.2. If both of the function blocks are located in one device, handling would be identical.

Both sets of variables are updated, even if only one set is displayed. When switching between the players, all values are available immediately.

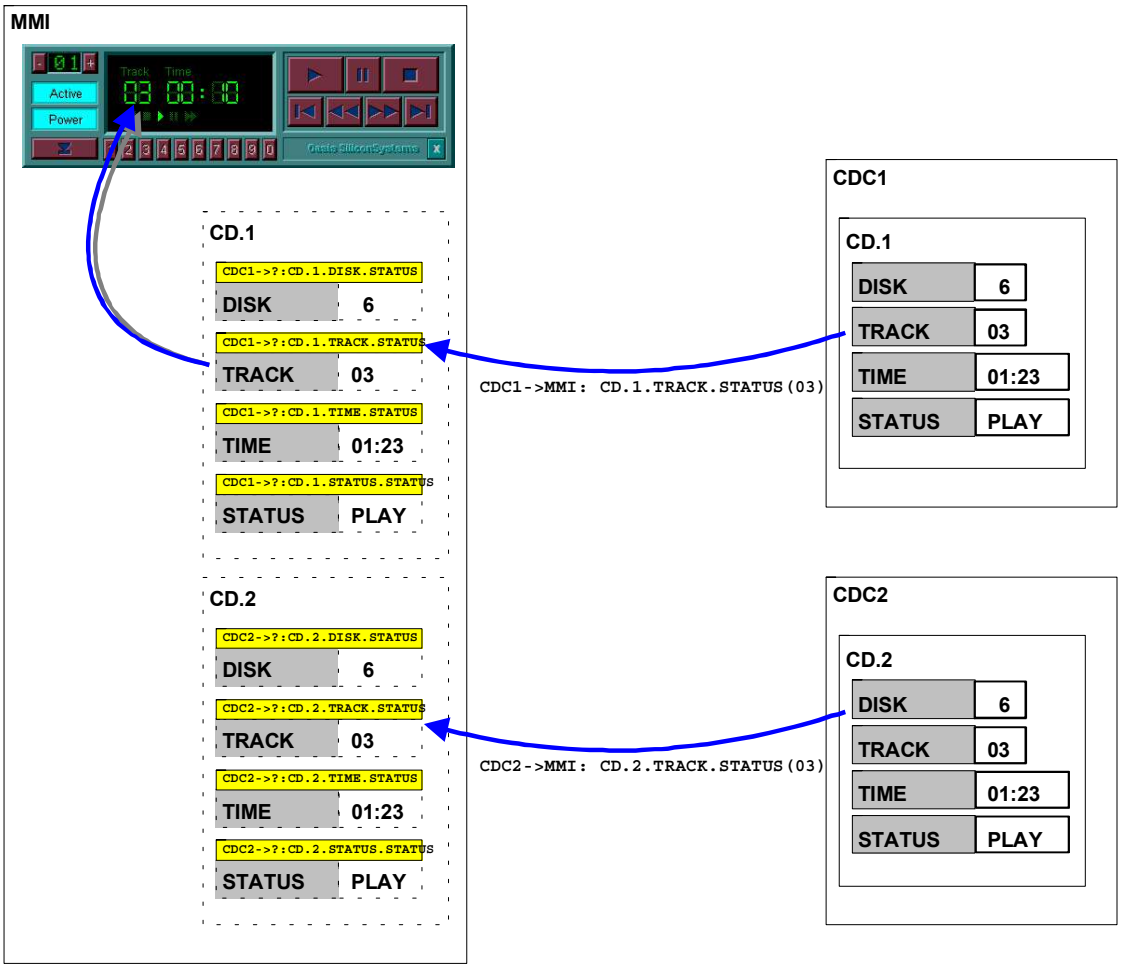


Figure 2-25: Controlling two identical devices.

For the assignment of protocols and variables in the control device, the respective protocols are defined for each variable. Each variable therefore has a filter function that can be passed only by the "own" protocol.

This can be done by a table, which contains the protocols consisting of MOST sender address, FBlockID, InstID, and FktID. There can be one pointer assigned to each protocol, pointing to the respective variable. In addition to that, or as an alternative, function pointers are also allowed. By this, functions could be called depending on protocols, and controlled by tables.

The concept can also be realized by an object-oriented approach, where variables are realized by objects with protocol filters and methods for representation. Following this approach, all incoming protocols are distributed to all objects, but only that object whose filter lets the protocol pass, will react. Analogously, the incoming protocols are compared to all protocols in the table when using the table approach.

On more complex control devices, this approach can be optimized by filtering the protocols step by step. The figure below shows an HMI which contains Shadows of a CD player and a tuner. These Shadows are implemented as interface objects. The interface objects are combined in two parent objects that filter the incoming protocols by sender address and InstID. The interfaces themselves only need a filter for the FktID.

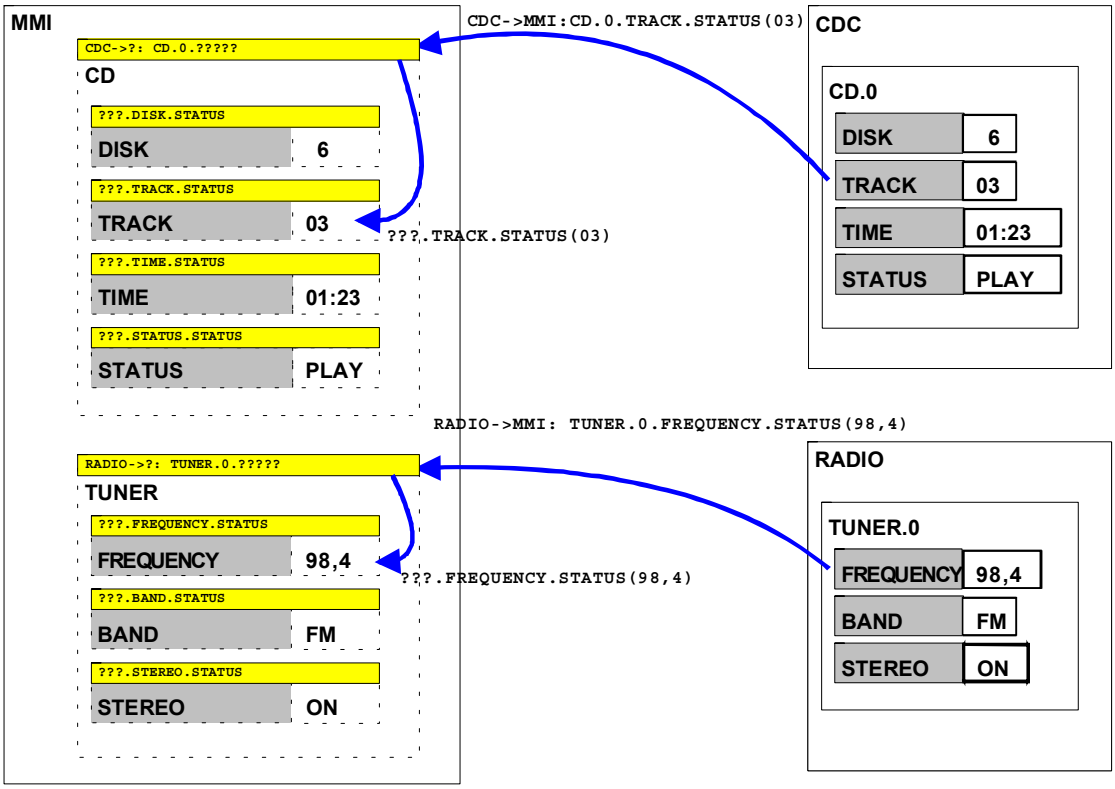


Figure 2-26: Hierarchical structure of the protocol filter (command interpreter).

Without object-oriented programming, the stepwise filtering can be implemented by using a message dispatcher. This dispatcher would forward the protocols to the respective function blocks based on sender address, FBlockID and InstID. Every function block can then analyze the FktIDs itself, by an own command interpreter.

Based on the well structured protocols, further analyzing steps can be inserted if required.

2.3.6.2 Communication With Methods

2.3.6.2.1 Standard Case

In general, communication with properties is equal to communication with methods. This means that a controller controls a function in a Slave Device and there will be a reply to the Controller Device. An example:

```
Controller -> Slave: FBlockID.InstID.StartResult  (Data)

Slave -> Controller: FBlockID.InstID.Result  (Data)

Slave -> Controller: FBlockID.InstID.Error  (ErrorCode, ErrorInfo)
```

2.3.6.2.2 Special Case Using Routing

In some cases there are methods where the general way of communication is not sufficient. The philosophy of building Shadows when on handling properties is based on the fact, that every property has only one single and unique state. This state then is imaged on one or more controllers. This condition is not valid for methods. So it may happen that a method is processing a request for one Controller, while it appears to another Controller to be busy. It has many states.

In addition to that, in methods a process is triggered, which has a longer processing time. The Controller might need to wait for a result. If several tasks within a device accessed one method at the same time, it must be possible to route the answer back to the respective task.

One example can be the SMS service in a GSM module of the device Telephone. In HMI, three tasks desired to send an SMS message independently from each other. The message of task 1 was sent, the one of task 2 was buffered, while the message of task 3 was rejected. The respective status message must now be assigned, which is not possible using the communication methods described up to now.

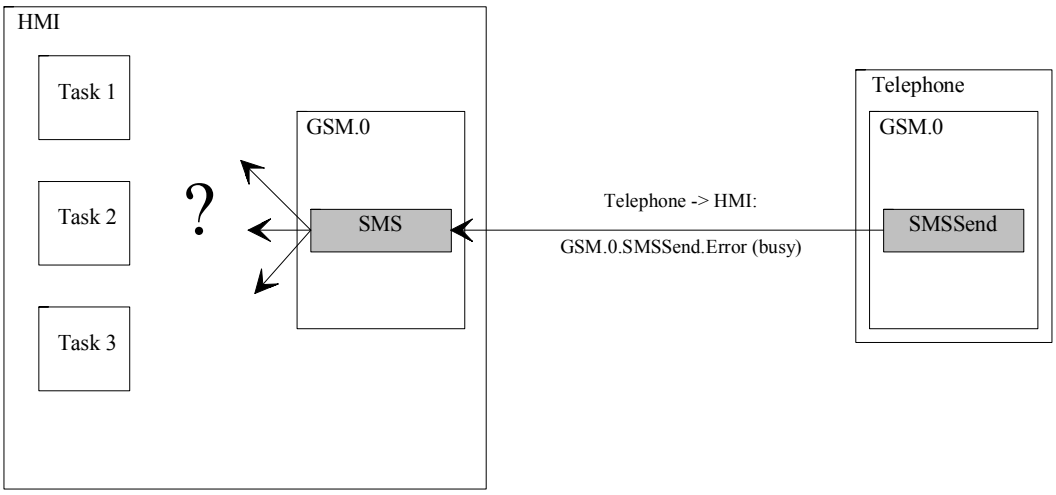


Figure 2-27: Routing answers in case of multiple tasks (in one controller) using one function.

To provide routing in such cases, the OPTypes StartResultAck, ProcessingAck, ResultAck, and ErrorAck are introduced. The behavior of these OPTypes is identical to that of StartResult, Processing, Result and Error. The only difference is, that as first parameter the SenderHandle (data type unsigned word) is inserted. The SenderHandle is set by the Controller at StartResultAck and characterizes the sender more in detail (Task, process...). The SenderHandle will not be interpreted by the Slave, but will be returned in an answer (ProcessingAck, ResultAck or ErrorAck).

The SMS call in Task 1 might look like:

```
Controller -> Slave: Telephone.0.SMSSend.StartResultAck
(SenderHandle1.SMSData)
```

After successful transmission, Task 1 gets:

```
Slave -> Controller: Telephone.0.SMSSend.ResultAck (SenderHandle1)
```

If Task 3 desires to send in the meantime, it sends:

```
Controller -> Slave: Telephone.0.SMSSend.StartResultAck (SenderHandle3.SMSData)
```

And it then gets in return:

```
Slave -> Controller: Telephone.0.SMSSend.ErrorAck (SenderHandle3.ErrorCode="Busy")
```

It must be decided individually, which methods must have a detailed back addressing with OPTypes StartResultAck, ProcessingAck, ResultAck, and ErrorAck.

2.3.7 Seeking Communication Partner

It may happen that an application has to seek a communication partner, that is, a function block. This might happen in a self-configuring audio system with four or six active speakers. The audio controller knows that function blocks with the FBlockID AudioAmplifier must be available, but does not know how many, or where. Therefore it has to seek, and gets the instance IDs as reply. With the help of the InstIDs and the number of audio amplifiers, it can configure itself correctly.

To seek a function block, the seeking block sends the following protocol to the NetworkMaster:

```
control -> ??? : NetworkMaster.CentralRegistry.Get ( FBlockID )
```

The NetworkMaster contains the central registry, which represents an image of the physical and logical system configuration. It answers with a list of all matching entries of the central registry with physical and functional address:

```
??? -> control : NetworkMaster.CentralRegistry.Status (
                 Rx/TxLog.FBlockID.InstID,
                 Rx/TxLog.FBlockID.InstID,...)
```

Optionally, the InstID can also be specified, to search for a certain function block:

```
control -> ??? : NetworkMaster.CentralRegistry.Get ( FBlockID.InstID )
```

If the respective function block does not exist, the NetworkMaster replies with an error and error code 0x07 “Parameter not available”. It returns the number of the parameter (0x01 in this case) and the value (FBlockID.InstID in this case).

2.3.8 Requesting Function Block Information from a Device

To obtain information about the function blocks contained by a device, every NetBlock has the property **FBlockIDs** (0x000). It will be read in the following way:

```
control -> ??? : NetBlock.FBlockIDs.Get
```

and answers with a list of the contained FBlockIDs. The function block that most characterizes the device (e.g., Tuner in a radio device) is listed first. The NetBlock does not need to be listed, as it is a mandatory function block in every device:

```
??? -> control : NetBlock.FBlockIDs.Status (FBlockID1.InstID1,
                                            FBlockID2.InstID2...
                                            FBlockIDN.InstIDN)
```

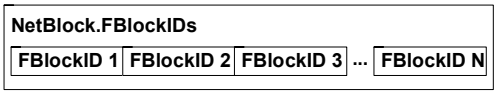


Figure 2-28: reading the function blocks of a device from NetBlock

2.3.9 Requesting Functions from a Function Block

In an adaptable system it may happen that a controller does not know exactly which functions are available in a function block (e.g., simple or high-end audio amplifier). Therefore, every function block has the function **FktIDs** (0x000). It is read as follows:

```
control -> ??? : FBlockID.InstID.FktIDs.Get
```

Within a function block, FktIDs between 0x000 and 0xFFF (4096 different FktIDs) can be available. The FktIDs are assigned as described in 2.3.2 on page 32. This raises the problem of a compact response, if the functions contained in a function block are requested. It is solved by a mechanism derived from the run length encoding. A bit field is built where the first bit is set to 1 if FktID 0x000 is available, the second bit is set to 1 if FktID 0x001 is available, and so on. Such a bit field might look like:

FktID		000	001	002	003	004	005	006	...	021	022	023	024	...	A00	A01	A02	A03	...	FFF
Bit field	1	1	1	0	0	1	1	0	0	0	1	1	0	0	1	1	0	0	0	0

The answer lists only the positions (FktIDs) where the bit state changes, beginning with an initial bit state of 1.

For the example shown above, the result would be:

```
??? -> control: FBlockID.InstID.FktIDs.Status (002 004 006 022 024 A00 A02 0)
```

The last 0 represents a stuffing nibble.

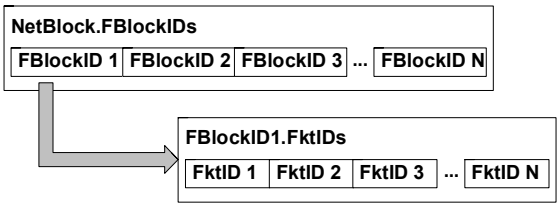


Figure 2-29: Requesting the functions contained in an application block.

2.3.10 Transmitting The Function Interface

2.3.10.1 Principle

In principle, function interfaces can be transmitted to a controller, or a HMI.

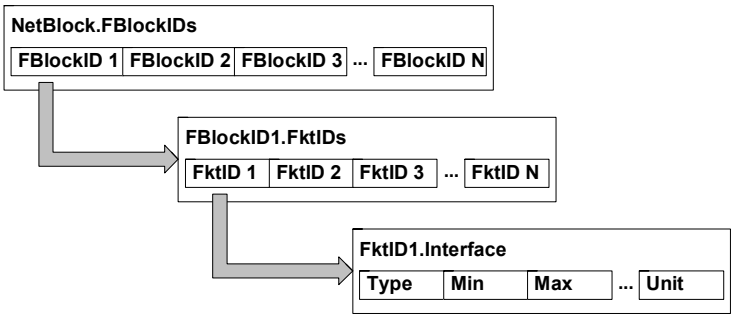


Figure 2-30: Requesting the function interface of a function

The flow for determining all function interfaces of a function block looks like:

```
control -> slave : FBlockID1.FktID1.GetInterface
slave -> control : FBlockID1.FktID1.Interface ( [ Interface Description ] )
control -> slave : FBlockID1.FktID2.GetInterface
slave -> control : FBlockID1.FktID2.Interface ( [ Interface Description ] )
...
control -> slave : FBlockID1.FktIDN.GetInterface
slave -> control : FBlockID1.FktIDN.Interface ( [ Interface Description ] )
```

The parameter list “Interface Description” contains information about a function interface.

2.3.10.2 Realization Of The Ability To Extract The Function Interface

In the function catalog, every interface of classified functions is described. By doing this, a classified definition of application protocols, as well as a uniform description, is possible, which can be based onto a few classes.

2.3.11 Function Classes

When having a look at function classes, properties and methods must be differentiated. The properties themselves consist of such with one variable, and of such with multiple variables.

2.3.11.1 Properties With A Single Variable

Many functions contain only a single variable. These functions can be divided into classes, which corresponds with the type declaration in programming languages. The class of a property is derived from the basis data type (Refer to section 2.3.2.7 on page 47) of its variable.

At the moment there are the following function classes for single variables:

Function class	Explanation
Switch	Properties of this class contain a variable of type Boolean (on/off; up/down). It can be set (Set) or read (Get, Status).
Number	Properties of this class contain a numeric variable (frequency, speed limit, temperature), which can be displayed (Status), set absolutely (set) or changed relatively (Increment, Decrement).
Text	Properties of this class have a string variable (Status), e.g., Warning, Hint.
Enumeration	Properties of this class contain a variable of type Enum. They provide an unchangeable number of invariable elements, from which can be chosen (Set). Examples: Drive status (Stop, Pause, Play, Forward, Rewind), Dolby (B, C, Off).
BoolField	Properties of this class contain a number of bits, that should either be used as flag field, or as controlling bits that are always manipulated together.
BitSet	Properties of this class are based on data type BitField. They contain a number of bits, which can be manipulated individually.

Table 2-9: Classes of functions with a single parameter.

The function classes (basic classes) with one variable, and their resulting protocols are described in detail below. The following universal parameters are used:

Flags: 8 Bit

D0 = 1	visible
D1 = 1	enabled
D2 = 1	UNICODE
D3 = 1	Notification

By using D0, the device can influence whether the function is displayed at the moment, or not (Default = 1). D1 may disable a function temporarily (like the gray options in PC application's menus). D2 shows whether a function uses Unicode or standard strings. D3 indicates whether a function supports notification (properties only).

Class:	8 Bit	0x00	Unclassified method	
		0x10	Unclassified property	
		0x11	Switch	
		0x12	Number	
		0x13	Text	
		0x14	Enumeration	
		0x15	Array	refer to section 2.3.11.2.2 on page 80.
		0x16	Record	refer to section 2.3.11.2.1 on page 78.
		0x17	Dynamic Array	refer to section 2.3.11.2.3 on page 83.
		0x18	Long Array	refer to section 2.3.11.2.4 on page 85.
		0x19	BoolField	
		0x1A	BitField	
		0xFF	Abort	(No further specifications behind this location)

OPTypes: 16 Bit — Bit field of available OPTypes (1 = OPType available). LSB represents the least significant OPType "Set", which has code 0x0.

Name: — Name of function as null terminated string.

2.3.11.1.1 Function Class Switch

OPType	Parameters
Set	Boolean
Get	
Status	Boolean
SetGet	Boolean
GetInterface	
Interface	Flags, Class, OPTypes, Name
Error	ErrorCode, ErrorInfo

Boolean: 1 Byte 0 for off, 1 for on

Example: RDSOnOff in AM/FMTuner1

```
Function:     RDSOnOff                  e.g. 0x00A

Flags:        visible, enabled,         0000 1011 = 0x0B
              no Unicode, notification

Class:        Switch                    0x11

OPTypes:      Get, SetGet, Status       1101 0000 0010 0110 = 0xD026
              GetInterface, Interface,
              Error

Name:         RDSOnOff                  "RDS"
```

Upload interface:

```
Tuner -> HMI: AM/FMTuner.1.RDSOnOff.Interface (0B 11 D026 "RDS")
```

Setting RDS = OFF:

```
HMI -> Tuner: AM/FMTuner.1.RDSOnOff.SetGet (00)
```

Please note:
This is a hypothetical example. It does not necessarily follow the MOST Function Catalog.

2.3.11.1.2 Function Class Number

OPType	Parameters
Set	Number
Get	
Status	Number
SetGet	Number
Increment	NSteps
Decrement	NSteps
GetInterface	
Interface	Flags, Class, OPTypes, Name, **Units, DataType, Exponent, Min, Max, Step**
Error	ErrorCode, ErrorInfo

DataType: uns. Byte — Type of variable:
0x00 Unsigned Byte
0x01 Signed Byte
0x02 Unsigned Word
0x03 Signed Word
0x04 Unsigned Long
0x05 Signed Long

Exponent: Signed Byte — Position of decimal point; Value = Number * $10^{Exponent}$

Min: Minimum value of variable of type *DataType*

Max: Maximum value of variable of type *DataType*

Step: Step width for adjusting type *DataType.* The following condition must always be true:
Max = Min + (n * Step)

NSteps: uns. Byte — Number of steps, as defined under "Step width for adjusting". Default value is 1, value 0 is not allowed.
NSteps has no exponent, but has the same unit like the Number parameter.

Units: uns. Byte — Unit

Unit	Encoding
none	0x00
Distance:	
cm	0x01
m	0x02
km	0x03
mils	0x04
Time:	
us (Micro second)	0x10
ms (Millisecond)	0x11
s (Second)	0x12
min (Minute	0x13
h (Hour)	0x14
d (day)	0x15
mon (Month)	0x16
a (Year)	0x17
Frequency:	
1/min	0x20
Hz	0x21
kHz	0x22
MHz	0x23
Volume:	
l (Liter)	0x30
gal (UK)	0x31
gal (US)	0x32
Consumption:	
l/100km	0x40
mils/gal	0x41
km/l	0x42

Unit	Encoding
Speed:	
km/h	0x50
mils/h	0x51
m/s	0x52
Temperature:	
°C	0x60
F	0x61
Volume:	
dB	0x70
Voltage:	
mV	0x80
V	0x81
Current:	
mA	0x90
A	0x91
Angle:	
Degrees	0xA0
Minutes	0xA1
Seconds	0xA2
$360°/ 2^{32}$	0xA3
$360°/ 2^{a}$	0xA4
Resolution:	
Pixel	0xB0

Table 2-10: Available units

2.3.11.1.3 Function Class Text

OPType	Parameters
Set	String
Get	
Status	String
SetGet	String
GetInterface	
Interface	Flags, Class, OPTypes, Name, **MaxSize**
Error	ErrorCode, ErrorInfo

MaxSize: uns. Byte maximum length of string

2.3.11.1.4 Function Class Enumeration

OPType	Parameters
Set	Pos
Get	
Status	Pos
SetGet	Pos
Increment	NSteps
Decrement	NSteps
GetInterface	
Interface	Flags, Class, OPTypes, Name, **Size, Name1, Name2,...**
Error	ErrorCode, ErrorInfo

Size: uns. Byte Length of enumeration
0 = no element
1 = one element
2 = two elements....

Name x: Null terminated string, representing the name of element x

Pos: uns. Byte Number of active element, or of element to be activated

Please Note:
Increment and Decrement must be interpreted like Predecessor and Successor in common programming languages.

2.3.11.1.5 Function Class BoolField

OPType	Parameters
Set	Content
Get	
Status	Content
SetGet	Content
GetInterface	
Interface	Flags, Class, OPTypes, Name, DataType, **NElements, BitName, BitSize, BitName, BitSize, ...**
Error	ErrorCode, ErrorInfo

Content: uns. Byte / uns. Word / uns. Long — Data area, containing e.g. flags

NElements: uns. Byte — Number of Elements in the BoolField

BitName: String — Null terminated string, indicating the name of the respective element

BitSize: uns. Byte — Number of Bits required for encoding the element. Encoding starts at the LSB.

If a variable of Class BoolField is defined, a field of either 8bits, 16bits, or 32bits will be reserved. Using the flags starts at the LSB. The value 0b**** ***0 means false and 0b**** ***1 means true. Manipulating a BoolField always requires the writing of the entire variable.

Example:
This example shows a BoolField based on "unsigned Word". There are 11 bits used for representing some flags. Please note, that it is also possible to combine several bits for representing a special element (flag).

	B	Y	T	E		1	
D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0
					F. 10	F. 9	F. 8

	B	Y	T	E		0	
D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0
F. 7	F. 6	F. 5	F. 4	F. 3	F. 2	F. 1	F. 0

2.3.11.1.6 Function Class BitSet

OPType	Parameter
Set	SetOfBits
Get	
Status	SetOfBits
SetGet	SetOfBits
GetInterface	
Interface	Flags, Class, OPTypes, Name, **Size**
Error	ErrorCode, ErrorInfo

Size: uns. Byte Size of SetOfBits (Mask + Data) in bytes

SetOfBits: BitField

BitSet in Arrays and Records:

A BitSet represents one variable. That means it is addressable as an entity via one dedicated value of Pos.

Example:

```
MyArray = Array of BitSet:    XXXX XXXX,0100 1001
                              XXXX XXXX,1110 0011
                              XXXX XXXX,0010 1101
                              XXXX XXXX,0111 1111
```

Requesting Status report (1):

MyArray.Get (PosX=0x0)

Answer:

```
MyArray.Status (PosX=0x0,     XXXX XXXX,0100 1001,
                              XXXX XXXX,1110 0011,
                              XXXX XXXX,0010 1101,
                              XXXX XXXX,0111 1111)
```

Requesting Status report (2):

```
MyArray.Get (PosX=0x2)
```

Answer:

```
MyArray.Status (PosX=0x2,     XXXX XXXX,1110 0011)
```

Performing a Set operation (1):

```
MyArray.Set (PosX=0x0,        1000 0001,1000 0001,
                              1000 0001,1000 0001,
                              1000 0001,0111 1110,
                              1000 0001,0111 1110)
```

Result:

```
MyArray =                     XXXX XXXX,1100 1001
                              XXXX XXXX,1110 0011
                              XXXX XXXX,0010 1100
                              XXXX XXXX,0111 1110
```

Performing a Set operation (1):

```
MyArray.Set (PosX=0x4,        1111 0000,1000 0001)
```

Result:

```
My Array =                    XXXX XXXX,1000 1001
                              XXXX XXXX,1110 0011
                              XXXX XXXX,0010 1100
                              XXXX XXXX,1000 1110
```

2.3.11.2 Properties with Multiple Variables

Some functions contain multiple variables. Here the principle should be to only combine functions that are very similar in nature (e.g., Station name and PI). Variables that do not match like that should be modeled in separate functions (e.g., Station name and current frequency).

Functions with multiple variables can also be assigned to classes called array and record. In an array, variables are of the same type, in a record they are of different types. It is possible to build an array of records, or a record containing an array. Such “two dimensional” constructs are allowed.

More complex constructs whose dimension exceeds two (array of array of record, or a record with two arrays), are definitely not allowed. In addition to that, it is not allowed to reference other functions from within a function. This means that an interface description of a function must not reference the interface descriptions of other functions. A function must be described completely and independent of other functions.

2.3.11.2.1 Function Class Record

OPType	Parameters
Set	Position, Data
Get	Position
Status	Position, Data
SetGet	Position, Data
Increment	Position, NSteps
Decrement	Position, NSteps
GetInterface	
Interface	Flags, Class, Name, **NElements, IntDesc1, IntDesc2...**
Error	ErrorCode, ErrorInfo

In the interface description of a record, the OPTypes are omitted, since they do not contain relevant information. OPTypes are relevant only in basic types.

NElements uns. Byte Number of elements in Record

IntDescX are the interface descriptions of the single elements. Depending on the data type, one of the interface descriptions defined for the respective class can be inserted. Please note that here in case of elements, parameter Flags is not available. In case of OPTypes (internal OPTypes here) only Set, Get, Status, Increment, Decrement and Error can be used.

Please note:
IntDesc only represents a group of parameters. No referencing of other functions and their interface descriptions is done here!

Below, IntDesc is displayed with respect to the basic classes:

Class	IntDesc
Switch	Class, OPTypes, Name
Number	Class, OPTypes, Name, **Units, DataType, Exponent, Min, Max, Step**
Text	Class, OPTypes, Name, **MaxSize**
Enumeration	Class, OPTypes, Name, **Size, Name1, Name2,...**
BoolField	Flags, Class, OPTypes, Name, DataType, **NElements, BitName, BitSize, BitName, BitSize, ...**
BitSet	Flags, Class, OPTypes, Name, **Size**
Array	Class, Name, **NElements, IntDesc**

Position always consists of two bytes, and indicates what will be set, requested, or read in the record. The first byte (x) indicates the position of an element in the record. If the record contains an array (two dimensions), the second byte specifies the line in the array. On:

- x=y=0,
  The operation is related to the entire record.
- x=(Position of array in record) AND y>0,
  The operation is related to a “line” in the array.
- x=(Position of array in record) AND y=0,
  The operation is related to the entire array.
- x<>(Position of array in record) AND y=0,
  The operation is related to the respective element in the record.

Even if the record does not contain an array, the position consists of two bytes, but the second byte is not used in this case.

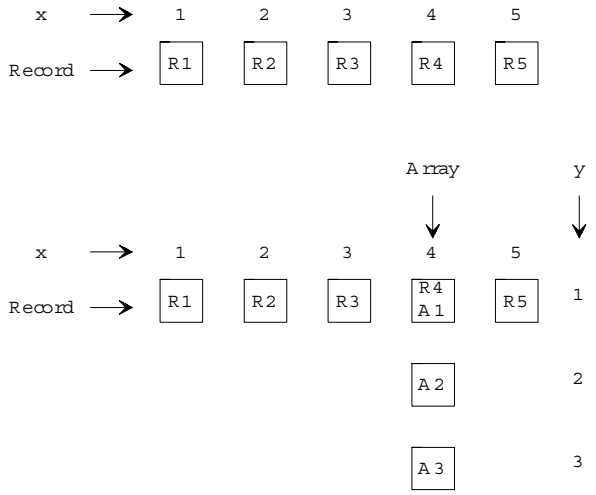


Figure 2-31: Meaning of position x in record (above) and of position y in a record with array (below).

Data represents data according to the structure of the record, and the specifications by position.

2.3.11.2.2 Function Class Array

OPType	Parameters
Set	Position, Data
Get	Position
Status	Position, Data
SetGet	Position, Data
Increment	Position, NSteps
Decrement	Position, NSteps
GetInterface	
Interface	Flags, Class, Name, **NMax, IntDesc**
Error	ErrorCode, ErrorInfo

Function class Array is very similar to Record. **NMax**, of type unsigned byte, represents the maximum number of elements. Since the array contains only elements of the same type, there only needs to be one IntDesc of the following type:

Class	IntDesc
Switch	Class, OPTypes, Name
Number	Class, OPTypes, Name, **Units, DataType, Exponent, Min, Max, Step**
Text	Class, OPTypes, Name, **MaxSize**
Enumeration	Class, OPTypes, Name, **Size, Name1, Name2,...**
BoolField	Flags, Class, OPTypes, Name, DataType, **NElements, BitName, BitSize, BitName, BitSize, ...**
BitSet	Flags, Class, OPTypes, Name, **Size**
Array	Class, Name, **NElements, IntDesc**
Record	Class, Name, **NElements, IntDesc1, IntDesc2...**

Analogous to the determinations of a record, the following is valid here for an array:

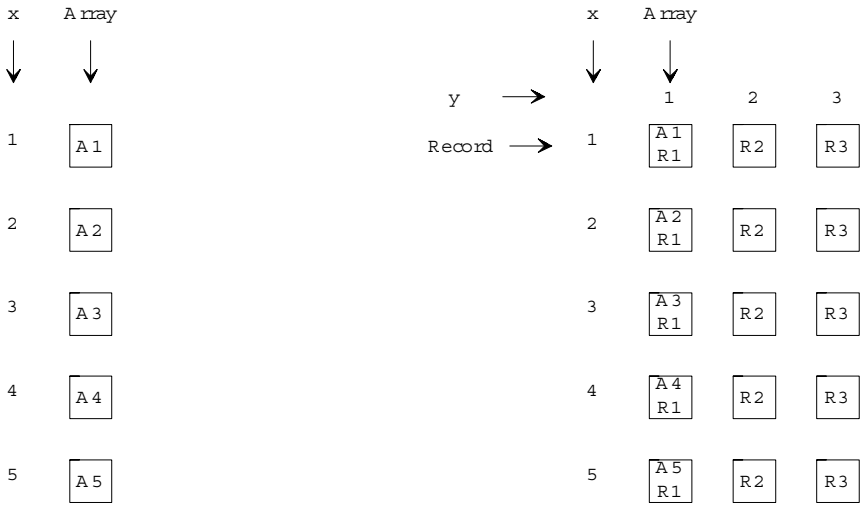


Figure 2-32: Position x in case of an array of basic type (left), and y in case of an array of record (right).

As in the case of a record, Position always consists of two bytes, independent of whether the array contains a record or not. If there is no record, the second byte is not used.

Please note:
The first parameter x (first byte) always refers to the outer structure, that is, the array for an Array of Record, and the record for a Record with Array.

If a partial structure is transmitted by using Position, the sending device is responsible for keeping consistency with the general structure transmitted before. As an example, the AM/FMTuner might update the signal qualities in a station list that was transferred earlier. It must take care to make sure that the signal quality values are assigned to the correct stations.

Transmitting an array is the only time when it is possible to transmit fewer elements than the maximum number of elements (NMax entry in the function interface FI). As an example, on 10 receivable stations the entire list of perhaps 100 possible entries does not need to be transferred. It must be kept in mind that each individual element of the array must always be transferred completely. If not, an error is assumed. The specification of the length is done in parameter Length of the application protocol.

If an array is empty, the status is reported without data:

```
FblockID.InstID.Array.Status (PosX=0x00, PosY=0x00)
```

Examples:

Disk information in CD changer:

The CD changer contains a magazine of up to 10 CDs. Each disk contains several tracks. The information is modeled in the two properties Magazine and Disk.

Magazine = Array[1..10] of Record of

DiskTitle:	String	(Text)
TotalTime:	Int	(Number)
NTracks:	unsigned Byte	(Number)

If a disk is not available, this can be recognized by TotalTime and NTracks containing 0x00. When requesting the FI, the formal answer is:

```
AudioDiskPlayer.0.Magazine.Interface

        (Flags. Class. Name. NMax.                                          | Array of
        Class. Name. NElements.                                             | Record of
        Class. OPTypes. Name. MaxSize                                       | Text
        Class. OPTypes. Name. Units, DataType, Exponent, Min, Max, Step     | Number
        Class. OPTypes. Name. Units, DataType, Exponent, Min, Max, Step)    | Number
```

or more related to the contents:

```
AudioDiskPlayer.0.Magazine.Interface

        (Flags. Array. "Magazine", 0A
        Record. "DiskInfo", 03
        Text. OPTypes. "DiskTitle", FF
        Number. OPTypes. "TotalTime". Seconds. Word. 00. 00 00. FF FF. 00 01
        Number. OPTypes. "Tracks". 00. Unsigned Byte. 00. 01. 63. 01)
```

On request

```
Controller -> CDC: AudioDiskPlayer.0.Magazine.Get (03. 01)
```

one receives the title of the third disk. On request

```
Controller -> CDC: AudioDiskPlayer.0.Magazine.Get (00. 01)
```

the titles of all disks are returned.

Disk = Array[1..99] of Record of

TrackTitle:	String	(Text)
TrackTime:	Unsigned Byte	(Number)

On requesting the FI, the formal answer is:

```
AudioDiskPlayer.0.Disk.Interface

        (Flags. Class. Name. NMax.                                                  | Array of
        Class. Name. NElements.                                                     | Record of
        Class. OPTypes. Name. MaxSize                                               | Text
        Class. OPTypes. Name. Units, DataType, Exponent, Min, Max, Step)            | Number
```

or more related to the contents:

```
AudioDiskPlayer.0.Magazine.Interface

        (Flags. Array. "Disk", 63
        Record. "TrackInfo", 02
        Text. OPTypes. "TrackTitle", FF
        Number. OPTypes. "TotalTime". Seconds. Word. 00. 00 00. FF FF. 00 01
```

Selecting In Arrays:

In many arrays, lines will be selected. Here, selections “1 of n” (one single line selected only) need to be differentiated from selections “n of N” (several lines can be selected at the same time).

- n of N:
  The selection here should be done by an individual parameter Selected of type Switch, which is used as prefix (Array of record of {Selected, ...}). The change in the status of the switch can be modified by Controller or Slave either single (Selected of a single line), or for an entire column (Selected of all lines). In principle this kind of selection can be used in case of 1 of N as well.

- 1 of N:
  In case of 1 of N there is an alternative modeling which is less expensive with respect to communication than n of N. Here a property Selected is modeled, which points onto the selected line. The kind of pointer differs individually. So e.g. in case of station lists the pointer may point onto the PI of the station currently active. In other cases, the position may be more effective. This way can be very effective, if a single line shall be selected in several Arrays (e.g. an entry in all telephone directories).

2.3.11.2.3 Function Class Dynamic Array

The arrays described above are optimized with respect to a high data volume. Navigation is based on the fixed sequence of elements in the array (Position = PosX, PosY). The position will not be contained in the data field. In Dynamic Arrays this is not possible, since here, lines can be inserted or removed (the sequence may vary). So a special function class DynamicArray is introduced, where PosX will be replaced by a uniquely defined handle, the Tag of data type uns. Word. It is defined as first parameter in the record:

```
DynamicArray = Array of Record of {Tag, ...}
```

For function class DynamicArray, the protocols are defined as follows:

OPType	Parameter
Set	Tag, PosY, Data
Get	Tag, PosY
Status	Tag, PosY, Data
SetGet	Tag, PosY, Data
Increment	Tag, PosY, NSteps
Decrement	Tag, PosY, NSteps
GetInterface	
Interface	Refer to section 2.3.11.2.2 on page 80
Error	ErrorCode, ErrorInfo

Tag	uns. Word	=	0x00 00	all lines
		<>	0x00 00	one special line
PosY	uns. Byte	<>	0x00	one special column (only if Tag <> 0x00 00)
		<>	0x01	not allowed, no access to Tag

Please note:
The Tag belongs to the data field. This means that it is returned at the start of every line. PosY = 0x01 denotes the Tag. With respect to consistency, accesses to a column are not reasonable. The last line in a Dynamic Array indicates the end. It starts with Tag 0xFFFF and contains dummy data. This line is included within the NMax counter.

Examples for positioning:

1. Array of Record of {Tag, El1, El2, El3, El4}
2. Tag = 0x00 00 and PosY = 0x00
3. Tag = 0x20 06 and PosY = 0x00
4. Tag = 0x6389 and PosY = 3

(1)

Tag	El1	El2	El3
0356			
3467			
3624			
2006			
0101			
6389			
0900			
3581			
9023			
FFFF			

(2)

Tag	El1	El2	El3
0356			
3467			
3624			
2006			
0101			
6389			
0900			
3581			
9023			
FFFF			

(3)

Tag	El1	El2	El3
0356			
3467			
3624			
2006			
0101			
6389			
0900			
3581			
9023			
FFFF			

(4)

Tag	El1	El2	El3
0356			
3467			
3624			
2006			
0101			
6389			
0900			
3581			
9023			
FFFF			

Editing In DynamicArrays:

Like in case of simple arrays, data contents can be modified by using Set. In many cases this is sufficient for DynamicArrays as well. Especially if the inserting and deleting of lines is done within the Slave only. If the inserting and deleting of lines is done by the controller as well, more complex editing functions are required. They will be defined as separate methods. Below there are two examples which are defined in a way, that they can be applied to several DynamicArrays (FktIDs), e.g. several telephone directories. So there is no need for an individual instance per array. So these functions will be placed in the range of Coordination (0x000..0x1FF).

By DynArrayIns (FktID=0x080), a number Quantity (uns. Word) of array elements (entire lines) will be inserted in DynamicArray FktID. The lines will be inserted after that line containing Tag. The data contents of the lines to be inserted will be transferred as Data.

```
DynArrayIns.Start (FktID, Tag, Quantity, Data)
```

DynArrayDel (FktID=0x081) deletes a number Quantity (uns. Word) of array elements (entire lines). This is performed starting at the element containing Tag, which is included within deletion.

```
DynArrayDel (FktID, Tag, Quantity, Data)
```

Examples:

DynArrayDel (FktID, 00 00, FF FF)	Deleting of entire array
DynArrayDel (FktID, 87 95, FF FF)	Deleting of entire array starting at line containing Tag 0x8795
DynArrayDel (FktID, 87 95, 00 01)	Deleting of the line containing Tag 0x8795
DynArrayDel (FktID, 87 95, 00 00)	No deleting

2.3.11.2.4 Function Class LongArray

A Slave transfers the arrays and DynamicArrays (as described above) to the registered controller using shadows. In case of changes, the shadows then will be updated. In case of big arrays that are changed very often, this may not be practicable any longer (Amount of memory in Controller, transmission time, bus load). Here another model – LongArray – must be applied. Where to place the boundary between LongArray and DynamicArray, is a matter of an individual decision.

The class LongArray consists of a function MotherArray and a function class ArrayWindow. It is possible to generate instances of class ArrayWindow dynamically. An ArrayWindow represents an extract, a window to the MotherArray. An instance of LongArray therefore consists of at minimum two functions, so LongArray is no simple function class.

2.3.11.2.4.1 MotherArray

The MotherArray is structured like a function of class DynamicArray. So communication of DynamicArray is identical to the communication of MotherArray, but there are only the OPTypes GetInterface, Interface and Error available (refer to section 2.3.11.2.2 on page 80).

The main difference compared to DynamicArray is, that the MotherArray is not controlled and viewed directly, but via one or more different functions. In the function interface of the MotherArray, all OPTypes are listed that can be executed via ArrayWindows. Below there is an example for a MotherArray as Array of Record of { Tag, Character, Number}:

Tag	El 1	El 2
6243	a	01
2100	b	02
5428	c	03
0101	d	04
3245	e	05
4562	f	06
0012	g	07
5342	h	08
9473	i	09
9343	j	0A
8367	k	0B
3752	l	0C
7698	m	0D
	.	
	.	
	.	
6354	x	1E
3425	y	1F
1045	z	20
FFFF	FF	FF

2.3.11.2.4.2 ArrayWindow

The ArrayWindow represents a part of the MotherArray. One main difference to other function classes is, that it is not useful to instantiate an ArrayWindow in a static way (via function catalog). In other function classes, where functions are instantiated in a static way, those functions describe fixed properties and methods of the Slave. Their state is identical for all controllers. With respect to its status, an ArrayWindow is strongly bound to a Controller. So there must be an individual ArrayWindow for each Controller. So it is possible that several HMIs have individual ArrayWindows to an address directory (MotherArray), which have different size and position.

So functions of class ArrayWindow are instantiated dynamically at runtime. Therefore a function block (that has a MotherArray, which shall be accessed by ArrayWindows), must provide a method CreateArrayWindow for instantiation, and a method DestroyArrayWindow (both of class Unclassified Method). The FktID is used as instance handle, which is transferred from Slave to Controller during instantiation. The FktIDs used here must be occupied in Layer II of the NetServices, since a dynamical extension of the command interpreter is not available.

Function	OPTypes	Parameter
CreateArrayWindow	StartResultAck	SenderHandle, FktIDMotherArray, PositionTag, WindowSize
	ResultAck	SenderHandle, FktIDArrayWindow
	ErrorAck	SenderHandle, ErrorCode, ErrorInfo
DestroyArrayWindow	StartResultAck	SenderHandle, FktIDArrayWindow
	ResultAck	SenderHandle
	ErrorAck	SenderHandle, ErrorCode, ErrorInfo

FktIDMotherArray		FktID of the MotherArray. It is not dynamic, since MotherArray is a property of class DynamicArray of the Slave
FktIDArrayWindow		FktID of the ArrayWindow. It is generated dynamically and represents the object handle which is transferred during instantiation. A range for such dynamically generated FktIDs is occupied in advance.
PositionTag	uns. Word	Top left corner of the ArrayWindow is positioned at PositionTag
WindowSize	uns. Byte	Number of elements contained by the ArrayWindow

The methods CreateArrayWindow and DestroyArrayWindow can instantiate and destroy ArrayWindows even of several MotherArrays. If e.g. in a telephone all telephone directories are available as MotherArrays, every HMI that is interested in a telephone directory may instantiate an ArrayWindow for the respective MotherArray. So it can be that e.g. three telephone directories may be watched by three ArrayWindows.

If a device enters sleep mode, all instances of ArrayWindows are destroyed. Every Controller stores the position of its ArrayWindow with the help of the Tag of the first line. During CreateArrayWindow, and by the help of Move (FktIDArrayWindow, Absolute, Tag), the window can be positioned again.

The status of an ArrayWindow is kept up to date in the Controller by using a shadow. Also in that case, it is the Slave's task to keep the shadow up to date. Here the notification mechanism of the NetServices cannot be used, since it is static. Notification for ArrayWindows is to be implemented at application level. For each ArrayWindow there is only one single shadow, which is located in the Controller that has instantiated it. The DeviceID of the Controller is transferred to the Slave during instantiation, so there is no need to implement a special notification mechanism for registering the controller.

By using the ArrayWindow, editing the MotherArray can be done in the conventional way:

```
ArrayWindow.SetGet (Tag, PosY, Data)
```

Tag	El 1	El 2
6243	a	01
2100	b	02
5428	c	03
0101	d	04
3245	e	05
5342	h	08
9473	i	09
9343	j	0A
8367	k	0B
3752	l	0C
7698	m	0D
	.	
	.	
	.	
6354	x	1E
3425	y	1F
1045	z	20
FFFF	FF	FF

0012	g	07
5342	h	08
9473	i	09
9343	j	0A
8367	k	0B

0012	g	07
5342	h	08
9473	i	B3
9343	j	0A
8367	k	0B

MotherArray ***ArrayWindow*** ***SetGet (9473, 03, B3)***

There is no function interface for an ArrayWindow, since it only represents a “view” onto the MotherArray. The MotherArray itself has a function interface that describes all operations that can be performed by using an ArrayWindow.

Function	OPType	Parameter
ArrayWindow	Set	Tag, PosY, Data
	Get	Tag, PosY
	Status	Tag, PosY, CurrentSize, AbsolutPosition, Data
	SetGet	Tag, PosY, Data
	Increment	Tag, PosY, NSteps
	Decrement	Tag, PosY, NSteps
	GetInterface	
	Interface	Refer to section 2.3.11.2.2 on page 80
	Error	ErrorCode, ErrorInfo

Tag	uns. Word	=	0x00 00 all lines
		<>	0x00 00 one special line
PosY	uns. Byte	<>	0x00 one special column (only if Tag <> 0x00 00)
		<>	0x01 not allowed, no access to Tag
CurrentSize	uns. Word	<>	current size of the MotherArray
AbsolutPosition	uns. Word	<>	absolute position of the Array Window in the Mother Array. The value specifies the position of the top left cell in the MotherArray and the counting starts at 0.)

2.3.11.2.4.3 Positioning An ArrayWindow On A MotherArray

Since an ArrayWindow represents an extract of the MotherArray, it must be positioned on the MotherArray in an appropriate way. Therefore two methods are defined. Method MoveAW is mandatory. An instance of MoveAW is used for all instances of ArrayWindows (FktID) of a function block.

```
MoveAW.Start (FktID, Mode, Number, Tag}
```

FktID		FktID of the ArrayWindow to be moved	
Mode	uns. Byte	00	Top
		01	Bottom
		02	Up
		03	Down
		04	Absolute

Top and Bottom:

Top and Bottom move the ArrayWindow to the start, or the end of the MotherArray respectively. The parameters Number and Tag are transferred as well, but they are unimportant.

Tag	El 1	El 2
6243	a	01
2100	b	02
5428	c	03
0101	d	04
3245	e	05
4562	f	06
0012	g	07
5342	h	08
9473	i	09
9343	j	0A
8367	k	0B
3752	l	0C
7698	m	0D
	.	
	.	
9643	w	1D
6354	x	1E
3425	y	1F
1045	z	20
FFFF	FF	FF

MotherArray

0012	g	07
5342	h	08
9473	i	09
9343	j	0A
8367	k	0B

ArrayWindow

6243	a	01
2100	b	02
5428	c	03
0101	d	04
3245	e	05

MoveAW.Start (FktID, Top, xx, xxxx)

9643	w	1D
6354	x	1E
3425	y	1F
1045	z	20
FFFF	FF	FF

MoveAW.Start (FktIDBottom, xx, xxxx)

Up and Down:

Up and Down are used for relative movement of the ArrayWindow, where the parameter Number (uns. Byte) defines the number of lines by which the ArrayWindow shall be moved. If the ArrayWindow has "hit" the start or the end of the MotherArray, no error will be reported, if Up or Down try to move it out of the valid range.

Tag	El 1	El 2
6243	a	01
2100	b	02
5428	c	03
0101	d	04
3245	e	05
4562	f	06
0012	g	07
5342	h	08
9473	i	09
9343	j	0A
8367	k	0B
3752	l	0C
7698	m	0D
	.	
	.	
	.	
6354	x	1E
3425	y	1F
1045	z	20
FFFF	FF	FF

MotherArray

0012	g	07
5342	h	08
9473	i	09
9343	j	0A
8367	k	0B

ArrayWindow

MoveAW.Start (FktID, Up, 03, xxxx)

0101	d	04
3245	e	05
4562	f	06
0012	g	07
5342	h	08

MoveAW.Start (FktID, Down, 05, xxxx)

9473	i	09
9343	j	0A
8367	k	0B
3752	l	0C
7698	m	0D

Absolute:

Absolute adjusts an ArrayWindow in a way, that the first line contains the desired Tag. If the Tag located too far by the end of the MotherArray, so that the ArrayWindow would exceed the valid range, the ArrayWindow will be placed like in case of using Bottom.

Tag	El 1	El 2
6243	a	01
2100	b	02
5428	c	03
0101	d	04
3245	e	05
4562	f	06
0012	g	07
5342	h	08
9473	i	09
9343	j	0A
8367	k	0B
3752	l	0C
7698	m	0D
	.	
	.	
	.	
6354	x	1E
3425	y	1F
1045	z	20
FFFF	FF	FF

0012	g	07
5342	h	08
9473	i	09
9343	j	0A
8367	k	0B

2100	b	02
5428	c	03
0101	d	04
3245	e	05
3245	f	06

MotherArray ***ArrayWindow*** ***MoveAW.Start (FktID, Absolute, xx, 2100)***

The second method SearchAW is optional. SearchAW provides a seeking of Searchstring in MotherArray through ArrayWindow (FktID). Search is performed in that element of each line, which is specified by PosY:

```
SearchAW.Start (FktID, PosY, Searchstring)
```

Seeking starts from the first line of ArrayWindow and runs down to the end of the MotherArray. Then seeking continues automatically at the start of the MotherArray and ends at the first line of the ArrayWindow. In case of success, the first line of the ArrayWindow is positioned onto the first line of the MotherArray which contains Searchstring. In case of failure, an error is reported (ErrorCode 0x07 “parameter not available”).

Tag	El 1	El 2
6243	a	01
2100	b	02
5428	c	03
0101	d	04
3245	e	05
5342	h	08
9473	i	09
9343	j	0A
8367	k	0B
3752	l	0C
7698	m	0D
	.	
	.	
	.	
6354	x	1E
3425	y	1F
1045	z	20
FFFF	FF	FF

0012	g	07
5342	h	08
9473	i	09
9343	j	0A
8367	k	0B

5428	c	03
0101	d	04
3245	e	05
4562	f	06
0012	g	07

MotherArray ***ArrayWindow*** ***SearchAW.Start (FktID, 02, „c“)***

2.3.11.3 Function Class For Methods

For methods there is only one function class, since methods may differ significantly with respect to the parameters transferred during Start and Result (in opposite to properties). Methods that have to transfer parameters in case of Start and/ or Result, belong to class “unclassified method” (0x00). They must be defined in a specific way.

The only function class for methods is function class Trigger. There are no parameters in case of Start/ StartResult, and it does not return parameters in case of Result or Processing.

OPType	Parameter
Start	
Processing	
Result	
StartResult	
StartResultAck	SenderHandle
ProcessingAck	SenderHandle
ResultAck	SenderHandle
ErrorAck	SenderHandle, ErrorCode, ErrorInfo
GetInterface	
Interface	Flags, Class, OPTypes, Name
Error	ErrorCode, ErrorInfo

2.3.12 Handling Message Notification

In many cases, HMIs and controllers must get information about values reaching their maximum, or about changes of properties in other function blocks. To avoid polling, events for automatic notification are defined. Such events must often be sent to several devices (e.g., two HMIs). Because of that, a notification matrix is implemented in every function block. The devices that should be notified of changes to the status of a function are registered in this matrix.

Please note:
Only properties can be admitted to the notification matrix!

Entry	Fkt 1	Fkt 2	Fkt 3	Fkt 4	Fkt 5
DeviceID1	x	x	x	x	x
DeviceID2		x		x	
free for entry					
free for entry					
free for entry					
free for entry					

Table 2-11: Notification matrix (x = notification activated)

The size of a notification matrix depends on the function block, on the number of properties, and on the number of device entries, each of which must be registered individually. The minimum number of device entries is three.

When taking into consideration that a DeviceID has 16bits, a FktID has 12bits, and that in some function blocks possibly all 64 possible nodes of the network must be registered, the notification matrix might be very big. Nevertheless, the following subjects should be kept in mind:

- The notification matrix is only a model. It does not dictate the software implementation method.
- Implementation might be done in very economical ways, e.g., by pointers in every function object that point to DeviceIDs.
- In most cases it is sufficient if the notification matrix has only a few entries.
- Group addresses are allowed as DeviceID in the notification matrix.

For very simple function blocks, for example, a CD changer, it is sufficient if the notification matrix provides only three entries for DeviceIDs. A very efficient implementation is possible. For example, by using a group address, all HMIs in the network can be notified of status changes.

Administration of the notification matrix is done via function **Notification**. If a controller desires to register, or to remove registration, it sends the following protocol:

```
Controller -> Slave: FBlockID.InstID.Notification.Set (Control, DeviceID,
                                                       FktID1, FktID2...)
```

The DeviceID of the controller is transported at the start of the protocol, as described in section 2.3.5 on page 53, but in order to enter group addresses, the DeviceID is transmitted in the parameter field as well. Parameter Control (8 bits, only 4bits used) specifies where the entry or deletion is done:

Control	Name	Comment
0x0	SetAll	Entry is done for all functions
0x1	SetFunction	Entry is done for the following functions (maximum is 4)
0x2	ClearAll	DeviceID of controller is deleted for all functions
0x3	ClearFunction	DeviceID of controller is deleted for the specified functions (maximum is 4)
rest	reserved	

Table 2-12: Parameter Control

On SetFunction and ClearFunction, at most 4 FktIDs can be specified (16 bits each), to avoid exceeding the maximum data length of 12 bytes of a MOST telegram.

In the table below, the protocols with the different controls for making entries in the notification matrix are listed together with the respective resulting entries.

Protocol	Entry	Fkt 1	Fkt 2	Fkt 3	Fkt 4	Fkt 5
Notification.Set (SetAll, DeviceID1)	**DeviceID1**	x	x	x	x	x
Notification.Set (SetFunction, DeviceID2, FktID2, FktID4)	**DeviceID2**		x		x	
	Free for entry					
	Free for entry					
	Free for entry					
	Free for entry					

Table 2-13: Protocols with different controls for making entries in the notification matrix, and the resulting entries.

Immediately after registration in the notification matrix, the controller receives the status reports of all functions it has activated as events. If a double registering occurs, that is, a device registers that has already been registered, the reports are sent as if the device has been registered for the first time. This also applies to registering with group addresses.

Deleting entries is done in a similar way.

If a controller desires to read information from the notification matrix, it sends:

```
Controller -> Slave: FBlockID.InstID.Notification.Get (FktID)
```

In general, all "Report" OPTypes (Status, Error, Interface) are notified. Status and (possibly) Error are reported spontaneously after registration. Interface is not reported directly after registration.

As an answer to this request, a list is returned that contains all DeviceIDs which activated the respective FktID:

```
Slave -> Controller: FBlockID.InstID.Notification.Status (FktID,DeviceID1,
DeviceID2,..DeviceIDN)
```

Please note:
In case of array properties, only those elements that have been changed are sent as status during notification.

Error handling:

If no more registering is possible, function Notification answers:

```
Slave -> Controller: FBlockID.InstID.Notification.Error (ErrorCode, ErrorInfo)
```

ErrorCode is 0x20 (Function specific) then, and ErrorInfo is 0x01 (Buffer overflow).

In case a controller registers at a time, where no valid values of the respective property are available, the property responds by ErrorCode 0x41 (not available).

Please note:
For keeping the system flexible, and for optimizing the communication effort with respect to the needs, the notifications are re-built at every system start (NetOn).

In case a property fails, i.e. the value is temporarily not available (e.g. sensor of a thermometer is broken), Notification sends the following message to all nodes that are registered for the respective property:

```
Slave -> Controller: FBlockID.InstID.Notification.Error  (ErrorCode = 0x41:
                                                          "Function not
                                                           available")
```

This message is also sent, in case a node registers for the property after the problem occurred.

3 Network Section

3.1 MOST Transceiver and its Internal Services

The MOST Transceiver OS8104 of Oasis SiliconSystems provides extensive tools for operating the MOST bus simply and safely, and for the transmission of data of different origins. Based on these tools, higher layers are defined. For a more detailed description of the MOST Transceiver please refer to [7]. The following sections give an overview of the features of the OS8104 that are available for simplifying the definition of higher layers.

3.1.1 Electrical Bypass (All Bypass)

If the bypass is closed, all signals received at the RX pin of the MOST Transceiver are electrically connected to the RX pin (shortcut). In this state, the respective device is “invisible” to the network. The device will be considered for the automatic counting of bus components only after opening the bypass, which gives access to the bus.

After the MOST Transceiver is reset, the bypass is closed. This allows a very fast startup of the system, especially on usage of an optical wakeup mechanism. The bypass must be opened by the controlling micro controller after wakeup of the component.

3.1.2 Source Data Bypass

In order to put data on the MOST network, the source data bypass must be opened in the device. That means that the data is no longer passed through the chip without being processed (that is, not handled by the routing engine RE), but can be routed now, e.g., from a source data port to the bus.

Based on the internal processing of data, a delay of two samples is added in the signal path. The source data bypass should be opened only in devices that put source data onto the bus on runtime.

3.1.3 Master/Slave, Active and Passive Components

Basically, a MOST system consists of up to 64 nodes with identical MOST Transceivers. By configuration, any of the transceivers can be the Timing Master, all the others are slaves. The Timing Master provides generation and transporting of system clock, the frames, and blocks. All Slave devices derive their clock from the MOST bus.

The Timing Master, as well as active Slave devices (source data bypass is open, device can put source data on the bus) add two samples of delay to the path of source data.

A passive Slave device has a closed source data bypass. Since in that case the routing engine is inactive, no delay is generated.

3.1.4 Data Transport

The bit stream is optimized in such a way that processing is easy and maximum functionality is supported. This includes mechanisms for automatic channel routing, network delay detection, and burst data channel management.

The MOST network technology defines an intelligent bit stream which is capable of providing all MOST network features as described above.

Data is transferred in a continuous bi-phase encoded bit stream yielding more than a 24.8Mbps data rate at a 44.1 kHz rate and a bit error rate of less than 10^{-10}.

Since the MOST system is fully synchronous, with all devices connected to the bus being synchronized to the bus, no memory buffering is needed (unlike isochronous, or asynchronous devices). This keeps cost low.

The sample frequency in a MOST system can be chosen in a range between 30kHz and 50kHz. The frequency depends directly on the application components. Some devices, for example, CD drives, work at a device-specific sample rate. In systems optimized for cost, such devices are regarded as fixed with respect to sample frequency. The sample frequency that is used most should be defined as the system frequency, to avoid sample rate conversion in the different devices.

3.1.4.1 Blocks

Organization of data transfer in blocks of frames is required for network management and control data transport tasks.

A block consists of 16 frames with 512 bits each. Per frame, 60 bytes of data are available for source data (synchronous and asynchronous packet data), while two bytes transport control data. The 2 bytes of 16 frames (1 block) are added to the control frame that transports a control telegram.

3.1.4.2 Frames

The MOST frame structure is designed in a way that provides maximum flexibility in terms of compatibility with a number of existing communication and data transport requirements without any drawbacks in implementation cost or processing overhead. It allows easy re-synchronization, clock and data recovery with the highest data quality and integrity. Built-in structures allow simple network management on the lowest layers avoiding overhead and cost shortcomings.

For synchronization, two different bus node types are required. A Timing Master that generates the frames, and Slave devices that synchronize to the Master clock on the bus.

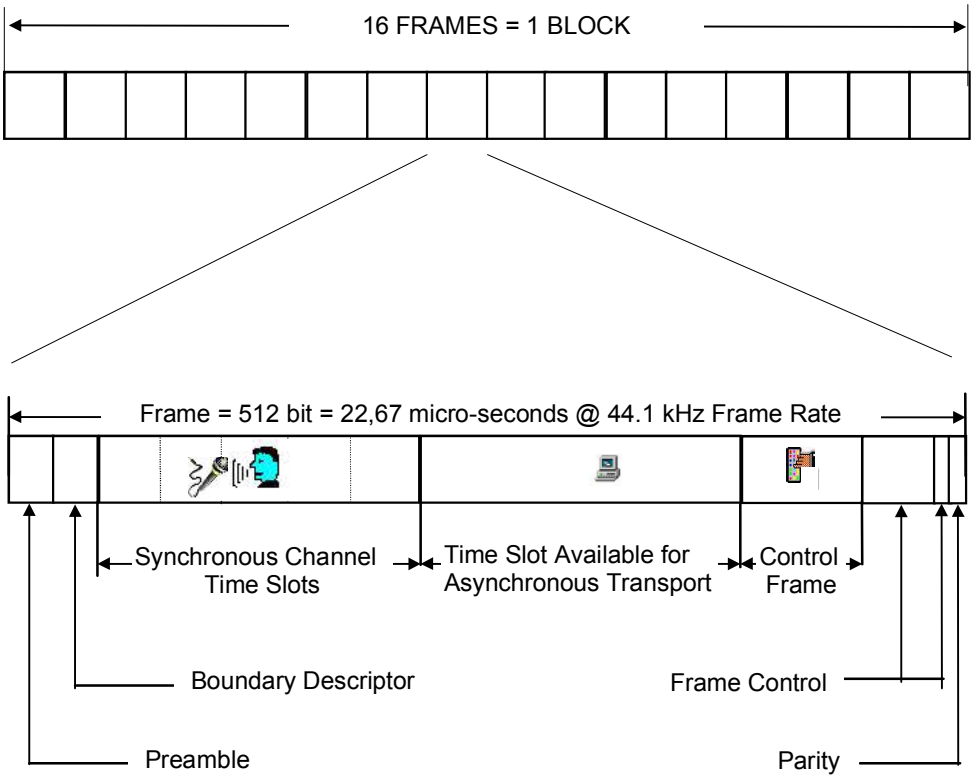


Figure 3-1: Structure of blocks and frames on the MOST bus.

The 64 bytes (512 bits) wide frame has the following structure:

Byte	Bit	Task
0	0-3	preamble
0	4-7	boundary descriptor (synchronous area count value)
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
63	511	parity bit

Table 3-1: Structure of the MOST frame

3.1.4.2.1 Preamble

The preambles are used internally to synchronize the MOST core and its on-chip functions to the bit stream.

For synchronization to a frame, two different mechanisms are used for Slave and Master nodes. For a Slave node, the first reception of valid preambles after reset, power-up, or loss of lock indicates that phase lock on the input bit stream has been accomplished.

This method ensures that the Slave node is phase- and frequency-locked to the bit stream, and hence the Master node. In a Master node, the transmitted bit stream is synchronized to an external timing source such as a crystal oscillator, SCK, FSY, or S/PDIF source.

Once all the nodes in the network have locked to the master's transmitted bit stream, the received bit stream has the correct frequency, but will be phase shifted with respect to the transmitted bit stream. This phase shift is due to delays from each active node, and additional accumulated delays due to tolerances in the phase lock within the Slave nodes. The Master node re-synchronizes the received data by the use of a PLL to lock onto the incoming bit stream, thereby re-synchronizing the incoming data to the proper bit alignment.

3.1.4.2.2 Boundary Descriptor

The boundary descriptor provides a flexible way of changing the bandwidth for synchronous and asynchronous data transmission. It represents the number of 4 byte blocks (quadlets) of data used for synchronous data. This value is used to determine the boundary between the synchronous and asynchronous data areas in the frame. A count value of 0 indicates no synchronous data and 15 quadlets of asynchronous data, while a count value of 15 indicates 15 quadlets of synchronous data and no asynchronous data.

By this means, a 60 byte data field can be allocated to either synchronous or asynchronous data on a 4 byte resolution. As such, it can be optimized to different requirements, depending on the amount of bandwidth required for each type of data.

Note that the maximum number of asynchronous data bytes per frame is 36 bytes, which means that the boundary descriptor values can be between 6 and 15.

The boundary descriptor is managed by the Timing Master of a MOST network. Please note that all synchronous connections must be re-built after having changed bSBC.

3.1.4.2.3 MOST System Control Bits

All other bits within the frame are for management purposes on the network level. While the preamble provides synchronization and clock regeneration, the parity bit indicates reliable data content and is used for error detection and phase lock loop operation.

3.1.4.3 Source Data

3.1.4.3.1 Definition of Control Data and Source Data

Depending on the kind of data and bandwidth, the MOST system provides different transmission procedures.

Telegrams for controlling devices or slow asynchronous data are transmitted via the control channel of the MOST Transceiver. For transmitting asynchronous data of higher bandwidth, a packet-oriented asynchronous data area is available. Synchronous data, such as audio signals of a CD drive, can be transmitted directly in the synchronous data area of the network. A more detailed description of the different data areas can be found in the sections below.

3.1.4.3.2 Differentiating Synchronous and Asynchronous Data

Sixty data bytes (15 quadlets) total are available for synchronous and asynchronous (packet) data. The number of synchronous and asynchronous bytes is specified by the boundary descriptor value described above.

3.1.4.3.3 Source Data Interface

The MOST Transceiver can handle a variety of different data formats at its source data port. The source data port formats are controlled via the internal registers of the transceiver. For more detailed information please refer to [7].

3.1.4.3.4 Transparent Channels

In addition to the different transmission procedures, the MOST network provides a transparent interface (transparent port). This port is oversampled (depending on the system's sample rate, and on the sampling rate chosen for the transparent port) and routed via the network. Therefore source data port 1 (SR1 and SX1) is available. It provides, for example, the transparent transmission of a RS232 interface, i.e., without synchronizing RS232 to the bus.

If no transparent channel is required, source data port 1 can be used as a standard source data port.

3.1.4.3.5 Synchronous Area

The synchronous channel time slots are available for real-time data such as audio/video or sensors and eliminate the need for additional buffering in analog-to-digital converters (and digital-to-analog converters) or in single speed CD devices for audio and video.

Accessing this data is provided by time division multiplexing (TDM) and allocation of quasi-static physical channels for a certain period of time (e.g., while playing an audio source). The bandwidth for such a channel can be adjusted by allocating any number of bytes to one logical channel. The maximum number of bytes available in a synchronous channel is 60 bytes/frame, which is corresponding to 60 x 8 bits or 15 stereo channels of CD-quality audio. The typical frame rate is 44,100 frames/second.

The routing engine (RE) is used to route data to and from the appropriate sources or sinks within a node. Internal synchronization is provided so input data does not need to be phase-aligned to the transceiver. The RE provides full flexibility in directing data from any source to any sink just by setting the appropriate value in the corresponding registers.

3.1.4.3.6 Asynchronous (Packet Data) Area

Another time slot is available for asynchronous data transport as required for more packet-oriented, burst-like data. In contrast to the control data channel, the asynchronous data channel provides transmission of longer data packets.

Access to this type of data is provided in a token ring manner. Each node has fair access to this channel and its bandwidth can be controlled using the boundary descriptor in a step of four bytes (quadlets). The maximum packet length on an asynchronous channel when using the 48 bytes data link layer, is 48 bytes. In case of using an alternative data link layer, the maximum packet length is 1014. The data on this channel is CRC protected. The asynchronous message is defined as follows:

Byte	Task
0	Arbitration
1-2	Target address
3	Length (in Quadlets = 4 Bytes)
4-5	Own address (Source address)
6-53	Data area
54-57	CRC

Table 3-2: Structure of a frame in the asynchronous area (48 bytes data link layer).

Byte	Task
0	Arbitration
1-2	Target address
3	Length (in Quadlets = 4 Bytes)
4-5	Own address (Source address)
6-1019	Data area
1020-1023	CRC

Table 3-3: Structure of a frame in the asynchronous area (alternative data link layer).

Since the asynchronous data area is variable, it can take several frames to complete a message. The corresponding management such as arbitration and channel allocation is provided by the chip. A hardware CRC is provided. The CRC is calculated in the background and can be indicated in a register at the end of each asynchronous message. A low level retry mechanism is not implemented.

3.1.4.4 Control Data

3.1.4.4.1 Control Data Interface

The transmission of control data to and from the MOST Transceiver is done via the control bus (at the control port). Depending on the application, this is either an I^2C bus, an SPI interface, or the parallel interface of the chip.

3.1.4.4.2 Description

The control data is used mainly for communication between the single nodes of the bus. This is where commands, status and diagnosis messages, as well as gateway messages are handled. The protocol on this channel runs in a carrier sense multiple access (CSMA) manner offering predictable response times which are considered essential in an audio/video control network. At a system sample frequency of 44.1 kHz, 2756 messages per second are transmitted, which corresponds to a gross data rate of 705.6 kBit/s.

Since 2 out of 64 messages are used for a system wide distributing of the allocation information by the network, the number of messages per second available for control messaging is 2670. When subtracting the data used for control and data securing, the net data rate (user data plus addressing) is 405.84 kBit/s, which corresponds to 19 bytes per message that can be read from the MOST Transceiver.

A MOST Device can access every third message propagated through the network. So a single device has a maximum message rate of 890 per second, or a net data rate of 135.28kBit/s

There are two kinds of control messages. Normal messages provide control of applications, while system messages handle system-related operations such as resource handling or remote access. During remote access, additional handshaking guarantees reliable remote operation. A control data message is 32 bytes long and has the following structure:

Byte	Task
0-3	Arbitration
4-5	Target address
6-7	Own address (Source address)
8-25	Data area+1 byte message type
26-27	CRC
28-29	Transmission status
30-31	Reserved

Table 3-4: Structure of a control data frame.

Please note:
The contents of register bXTIM – which controls the delay time between two messages in case of low level retries – must be identical in all nodes of a MOST Network.

One complete message is buffered by the chip. Any important status changes can be flagged by interrupt (if desired). The data on the control channel is protected by CRC and acknowledge mechanisms.

Arbitration is provided automatically by the transceiver in case a node wants to send a message. In order to provide fair arbitration even at high bus loads, a double arbitration mechanism is used. This ensures that an access is not depending on the communication load of upstream devices and the priority is not depending on the network position. Rejection of messages is flagged and automatic retransmission is performed. The number of retries can be defined by the application software register bXRTY. If the maximum of retries is reached without success, a transmission error is indicated to the controlling device (e.g., external micro controller). The delay time between the sending out of a message (in case of low level retries only) is specified in register bXTIM. The contents of bXTIM must be equal in every node of a MOST Network.

In standalone mode, the MOST control messages can be used to indicate the occurrence of an interrupt.

3.1.5 Internal Services

3.1.5.1 Addressing

The MOST Transceiver supports four different ways of addressing:

- Node position in the ring.
  The node position is generated automatically in each node during the locking procedure of the MOST network.
- Unique node address (2 bytes).
  This address can be set by the application. The SAI procedure helps to test on the chip level whether a desired address is unique or not.
- Group address (1 byte).
  Group address can be set by the application. A group is made up of devices that have the same number in the group address register.
- Broadcast
  The broadcast address is a special group address. When used, the message is received by all nodes in the ring. Until the last node in the ring has acknowledged a broadcast message, communication via the control channel is suppressed for other messages.

The different ways of addressing are mapped into the address area of a MOST Transceiver:

Address range	Mode
0x0001..0x02FF 0x0500..0xFFFF	Normal addressing (Point to point) based on unique node address.
0x0400..0x04FF	Node position addressing: Address = 0x400 + Node position of target node
0x0300..0x03C7 0x03C9..0x03FF	Group addressed: Address = 0x300 + Address of desired group Group address 0xC8 reserved for broadcast
0x03C8	Broadcast addressing

Table 3-5: Addressing modes vs. address range.

Group addressing is typically used for controlling several devices of the same type (e.g., active speakers). The grouping of devices must be established during definition of the system.

3.1.5.2 Address Initialization (SAI)

The MOST Transceiver supports a procedure on the chip level that can be used to verify whether an address is unique or not. The desired address must be written as a target address to the Xmit control message buffer (bytes XTAH and XTAL), and then the bit SAI in the message control register must be set to '1'. Now the transceiver checks whether the address is unique or not. If it is ok, then the address is immediately written to the node address registers of the transceiver. The result of the operation is written to bit TXR in the message status register, with a value of '1' meaning that the address was ok.

3.1.5.3 Support at System Startup

The MOST Transceiver meets all requirements of a low level startup. It is so far able to run in standalone mode, as it provides a communication-ready network to all applications. In addition, several supporting mechanisms are provided. All components of the system get a unique number, with numbering starting at the Timing Master at 0x00, and then incremented by one. These numbers can be used for node position addressing. Furthermore, every device receives the information about the total number of devices in the ring. The MOST Transceiver also provides a wakeup mechanism.

3.1.5.4 Delay Recognition

Based on the fact that every node may be active or passive with respect to source data handling (source data bypass open or closed), and that every active node generates two samples of delay, it is useful to have information about source data delay, for example, for noise compensation applications, or in high-end audio applications.

Therefore a mechanism is implemented in each MOST Transceiver, giving access to information about the total delay of the system, and to the delay up to the local node with respect to the Timing Master.

3.1.5.5 Remote-Access

This feature provides a remote-controllable I^2C bus in each MOST Transceiver that runs in standalone mode. It can be controlled from any node in the network, by using certain system commands. Furthermore, all registers on page 0x0 of the MOST Transceiver can be read or written via the network.

3.1.5.6 Automatic Channel Allocation

Since administration of up to 30 audio channels would need many resources on an application's side, the MOST system supports resource administration on chip level.

Allocating one or more audio channels (up to 64 bits per allocation procedure) is done via a request from an application to the Timing Master of the network. If there are enough channels, the application will get a handle, by which source data can be routed onto the network. The handle can also be used for de-allocating. A channel resource allocation table (mCRA), distributed automatically in the ring (on chip level), gives access to the current allocation status of the channels in each node.

The channel map that belongs to the handle can be retrieved from mCRA, or it can be delivered during connection management via control messages. The entire allocation takes about 25ms (maximum), making it possible to change allocation during runtime.

3.1.5.7 Power Management

The MOST Transceiver has three power states:

- Normal operation mode
- Low power mode
- Zero power mode

In normal operation mode, all sections of the chip are running and the chip is fully accessible.

Low power mode is used to lower the power consumption of devices that are not in use at the moment. All sections of the chip that handle source data are switched off, source data bypass is active. The chip is visible from the network's point of view, but no communication is possible. Based on the Timing Master, all components that are in low power mode can be awakened by a wakeup-preamble.

On zero power mode, all sections of the chip are deactivated, except a small wakeup logic. The wakeup logic activates the receiving FOT unit by a special pin of the chip in regular intervals and scans for modulated light. If there is modulated light, the chip wakes up.

When using an FOT unit with wakeup feature, the zero power mode of the chip does not need to be used. The chip can be connected to the switched branch of the power supply.

3.1.5.8 Detection of Unused Channels

Detection of unused channels, i.e., channels that are allocated by a device, but which are no longer used, is done with the help of the allocation table mCRA. By checking the most significant bit, it can be determined whether a channel is in use or not. If there are unused but allocated channels, they should be de-allocated with respect to the network resources.

3.2 Dynamic Behavior of a Device

3.2.1 Overview

This section describes the dynamic behavior of the system—the states and state transitions of the system, with a special focus on network dynamics (or the dynamic of the network interface of a device). The expression NetInterface stands for the entire communication section of a device, that is, the optical interface, MOST Transceiver, and NetServices.

The figure below shows a layer model of a device. The lowest layer is the power supply. On this layer, every hardware function is built, that is, the hardware of the NetInterface, which is made up of the MOST Transceiver, the optical interface, and the controller on which the NetServices are running. The NetServices make up the next layer, on which the higher services of address management, power management and network error management are based. At the top layer there is the application itself.

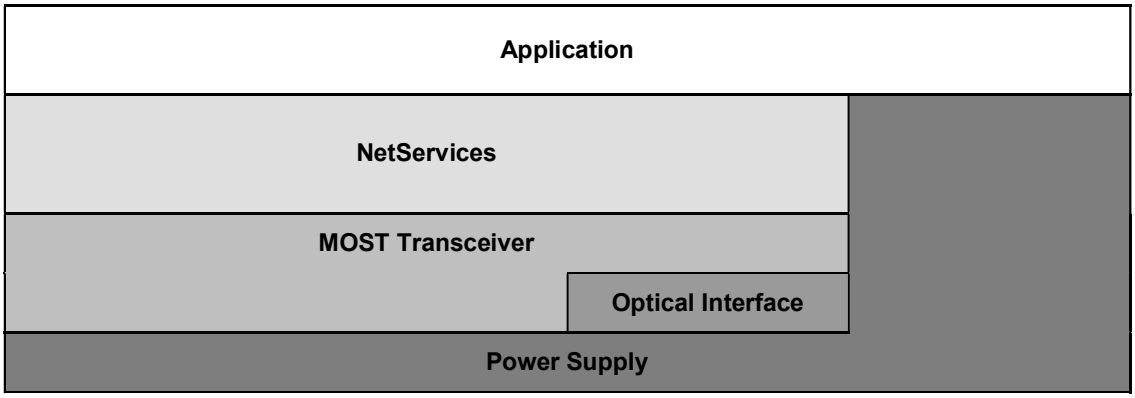


Figure 3-2: Layer model of a device

Generally, for each device, the device specification must define all the possible combinations of the states of the application section and the communication section. Especially from the view of the network, there are three states that are mandatory for each device:

1. **DevicePowerOff:** Communication section is in state NetInterfacePowerOff. The application section in a non-waking device is in state ApplicationPowerOff, or in a waking device in state ApplicationSleep.
2. **DeviceStandBy:** This state is mainly influenced by state ApplicationLogicOnly. The logical function of the application is running, while peripherals with high power consumption such as drives are switched off. This state is reached after state DevicePowerOff. The communication section is in state NetInterfaceNormalOperation.
3. **DeviceNormalOperation:** The communication section as well as the application section are in state NormalOperation.

Since these main states are only a few of all possible device states, it is not useful to use a diagram. The following description gives an impression of what might happen in the single states with respect to the communication and application sections.

DevicePowerOff:

- The application might be awakened, e.g., by a timer, can check an external signal, and return to state ApplicationSleep without waking up the NetInterface. The device does not leave the mode.
- The application might be awakened, e.g., by a timer, and can then wake up the NetInterface, and by that the entire network. The device changes to state DeviceStandBy, or state DeviceNormalOperation
- The application might be awakened by light on the bus and then wakes the application during initialization phase. The device changes to state DeviceStandBy.

DeviceStandBy:

- If the application is used, or its peripherals are in use, the device changes to state DeviceNormalOperation.
- If light on the bus is switched off, the device changes to state DevicePowerOff.

DeviceNormalOperation:

- If light on the bus is switched off, the device changes to state DevicePowerOff.

The following description of the dynamic behavior is done from the bottom up. The most significant subjects regarding power supply are described in section 4.1 on page 183. The following section focuses on the dynamic behavior of the NetInterface.

3.2.2 NetInterface

Here, the states of a device are seen from the view of the NetInterface. Operations within the application of a device are not considered. Only the interfaces to the application are shown. The following figure shows the states of the NetInterface and the events that lead to state transitions. The following sections explain the individual states.

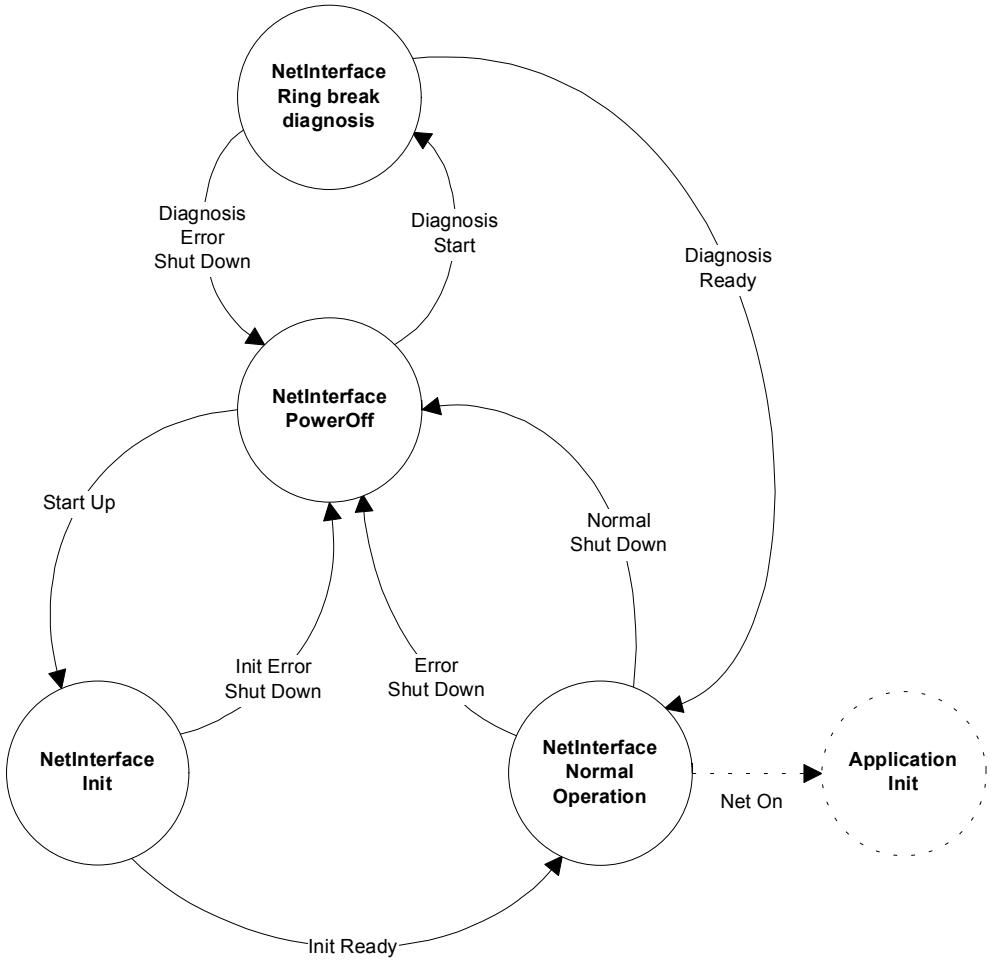


Figure 3-3: Flow chart "Overview of the states in NetInterface"

3.2.2.1 NetInterfacePowerOff

In state NetInterfacePowerOff, the NetInterface is switched off from the view of the network. The FOT does not emit light. The MOST Transceiver does not necessarily need to be switched off, since the application might still use function groups of the chip (e.g., RMCK generation).

State NetInterfacePowerOff is left when one of the following events occurs:

Event	Transition to	Cause
Start Up	NetInterfaceInit	A NetInterface is activated either by light at the receiving FOT, by the application (Hypothetical example: phone receives a call), or by a switch at the device.
Diagnosis Start	NetInterfaceRingBreakDiagnosis	A NetInterface is activated by connecting to power (for information about signal SwitchToPower please refer to section 4.1 on page 183)

Table 3-6: Events in state NetInterfacePowerOff

3.2.2.2 NetInterfaceInit

In this state, NetInterface is initialized to the point where the MOST Transceiver is able to communicate with other nodes.

This state is left when one of the following events occurs:

Event	Transition to	Cause
Init Ready	NetInterfaceNormalOperation	NetInterface is ready for communication (see below).
Init Error Shut Down	NetInterfacePowerOff	Error occurred during initialization (see below).

Table 3-7: Events in state NetInterfaceInit

Causes for event Init Ready:

- In the Master device:
  Net Activity and stable lock (at minimum for time t_{Lock}) were recognized. Lock is called stable if for a period of time t_{Lock} no unlock events occurred.
- In a Slave device:
  Stable lock (at minimum for time t_{Lock}) was recognized and the contents of register bSBC has a valid value (>5). This fact is the basis for the statement that the Timing Master of the system has also recognized stable lock, and the ring is closed.

Causes for event Init Error Shut Down:

- In the Master device:
  Timeout t_{Master}, occurs before a stable lock can be recognized.
- In a waking Slave device:
  Timeout t_{Slave}, occurs before a closed ring can be recognized. Error Error_NSInit_Timeout is stored by the application.
- In a non-waking Slave device:
  Timeout t_{Slave}. expires before light was recognized, or a closed ring was recognized; or the light was switched off again.

In a Slave device (non-waking) the bypass of the transceiver is deactivated (opened) as soon as a short lock is recognized (i.e., the lock does not need to be stable for t_{Lock}). In case of a waking Slave device and the Master device, the bypass is deactivated immediately after having entered this state (light at the output).

The flow chart below shows the behavior in state NetInterfaceInit. A differentiation is made between Master and Slave. On this level, Master means Timing Master and Slave means Timing Slave.

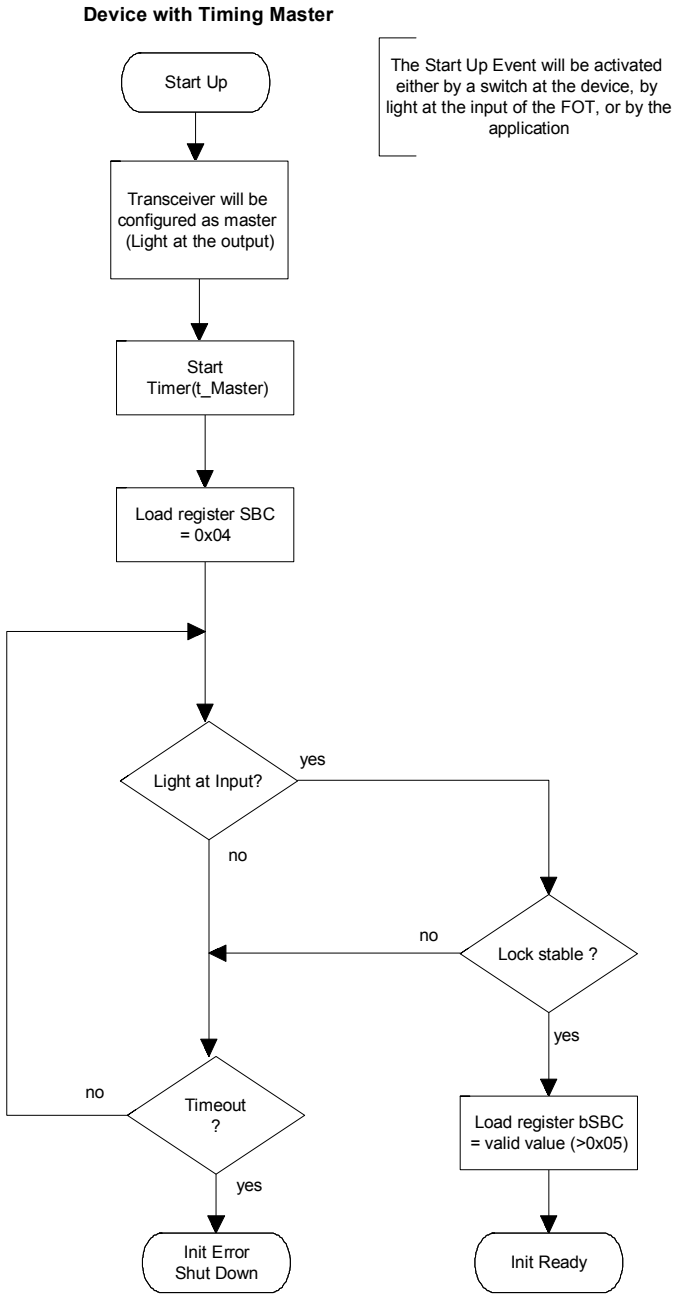


Figure 3-4: Behavior of a Master device in state NetInterfaceInit.

When entering state NetInterfaceInit, the Timing Master loads its bSBC register with the “invalid” value 0x04. This value is transferred to all transceivers via the frame. As soon as the Timing Master recognizes a stable lock, it loads its bSBC register with a valid value (>0x05). By doing this, every Slave in the ring can recognize when the Timing Master has reached stable lock.

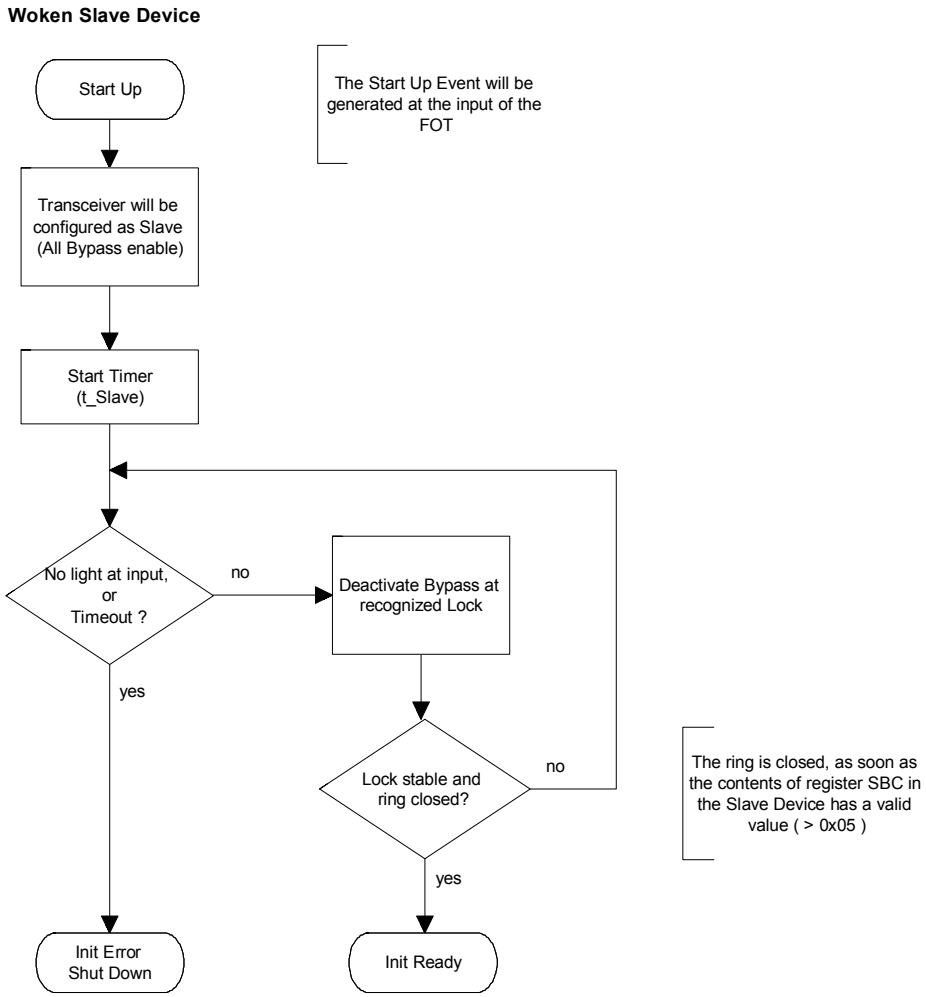


Figure 3-5: Behavior of a woken Slave device in state NetInterfaceInit.

Waking Slave Device

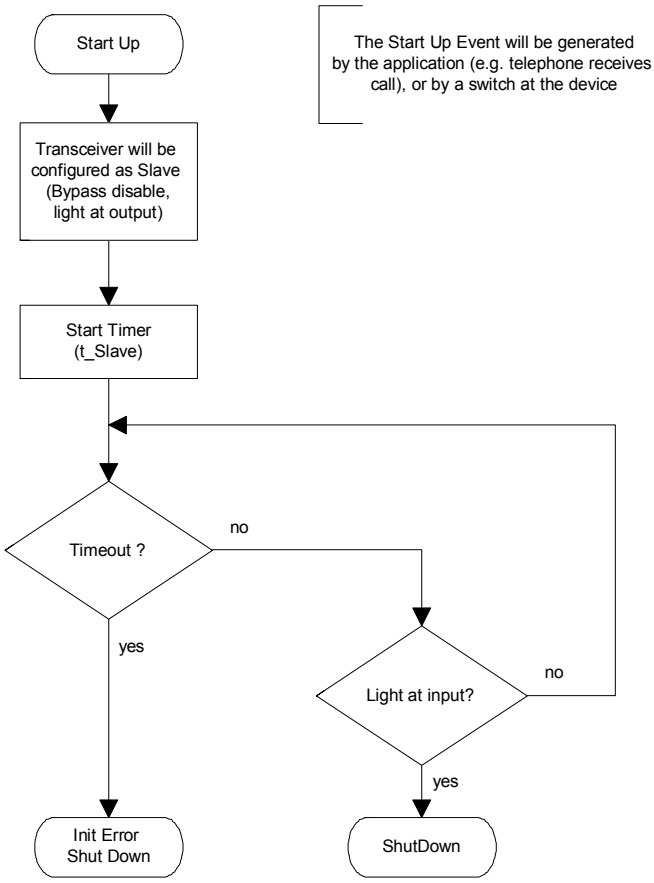


Figure 3-6: Behavior of a waking Slave device in state NetInterfaceInit.

After having woken the ring (the light returned from Timing Master), the Slave Device goes to ShutDown. From there it starts up as a standard Slave Device, woken by the Timing Master.

3.2.2.3 NetInterfaceNormalOperation

This state is reached as soon as the initialization has reached a level where the MOST Transceiver can start to communicate with other nodes in the network. When entering this state, the part of the application that is connected to the communication section is initialized.

Examples for initializing a higher layer due to Net On Event:

- Check of system configuration and building of the central registry (refer to section 3.3.5.3 on page 148).
- Setting of the logical node address and group address (refer to section 3.3.1 on page 144).
- Initialization of the sending and receiving units in the NetServices.

In certain circumstances, other application units are initialized earlier, independently from the state of the NetInterface.

Event	Transition to	Cause
Normal Shut Down	NetInterfacePowerOff	NetInterface will be deactivated by switching off light.
Error Shut Down	NetInterfacePowerOff	NetInterface will be deactivated due to a critical unlock.
Net On	Report to an application	Entering state NetInterface Normal Operation

Table 3-8: Events in state NetInterfaceNormalOperation

The Normal Shut Down event is generated as soon as no light is recognized at the input.

In state NetInterfaceNormalOperation, the NetServices check the lock state of the PLL of the MOST Transceiver. On an unlock, the application is informed as soon as possible by an unlock event. Every application then has to save its output signals (e.g., amplifier mutes its outputs).

In addition to that, the NetServices check the length of an unlock, or the occurrence of a series of unlocks. If the length of a single unlock exceeds the time t_{Unlock} , an Error Shut Down event (critical unlock) is generated.

If there are no further unlocks within time $t_{UnlockRecovery}$ before t_{Unlock} is exceeded, the lock state is regarded as stable. The application will get informed about this Lock Event, for restoring its signals.

In case a series of unlocks occurs, where the single unlocks do not exceed t_{Unlock}, but the interval of the occurrence is less than $t_{UnlockRecovery}$, an Error Shut Down event (critical unlock) is generated as well.

In addition to that, the MPR register is checked. If this register indicates a NetworkChange event, the application will get informed about that.

The flow chart below shows the behavior in state NetInterfaceNormalOperation:

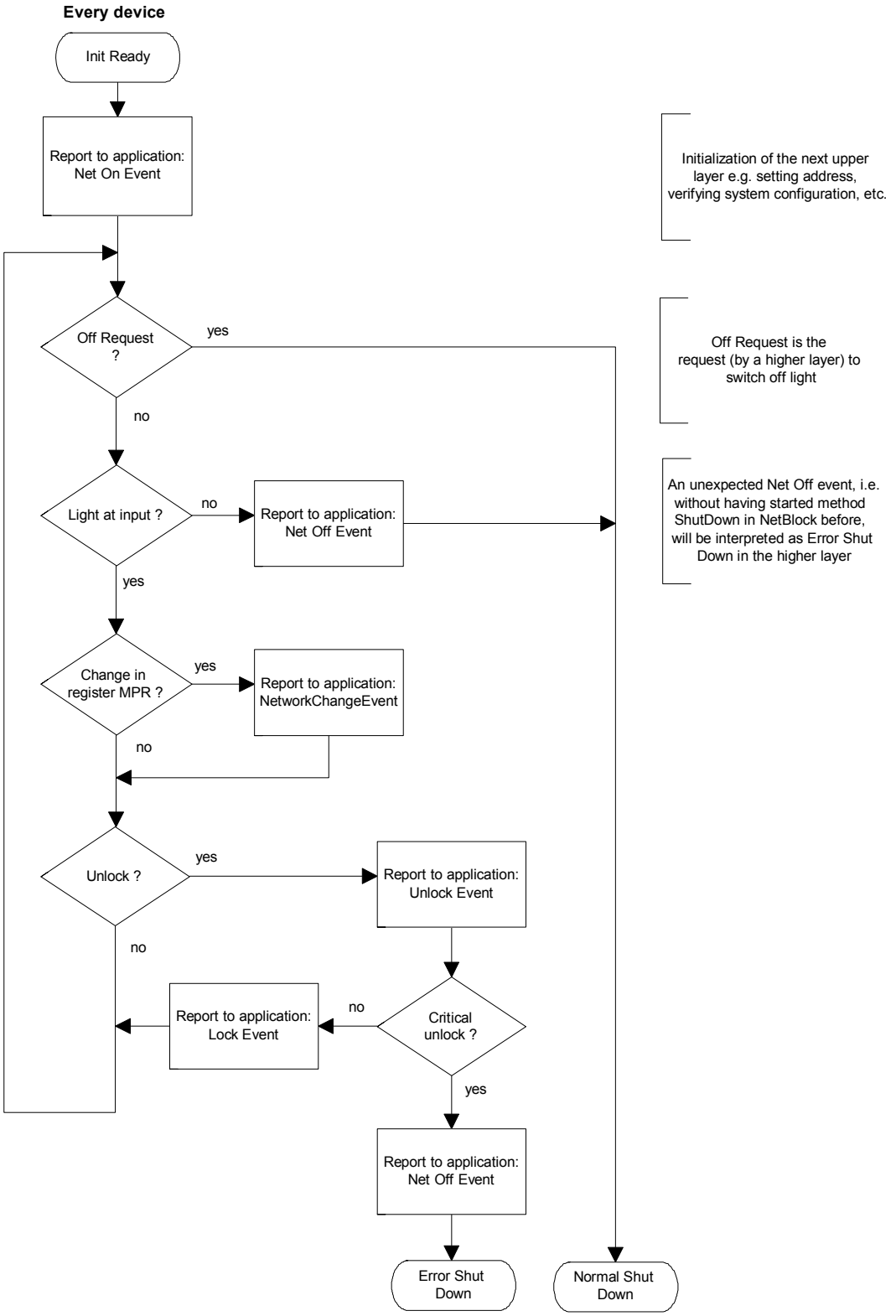


Figure 3-7: Behavior in state NetInterfaceNormalOperation.

3.2.2.4 NetInterface Ring Break Diagnosis

A simple recognition of a fatal error is possible in any state. Ring break diagnosis serves the purpose of localizing a fatal error in the network. It is run not during normal operation, but in the car repair, or at the assembly line.

The RingBreakDiagnosis process can be started by various triggers, which must be chosen and implemented by the System Integrator. One possible way is, to start the RingBreakDiagnosis by disconnecting the System from the power source for a short time. In this case, RingBreakDiagnosis is entered, when signal SwitchToPower of the SwitchToPowerDetector indicates, that the device was connected to power first time (e.g. after reconnection of the car's battery). This signal is not evaluated during NetOn.

In state NetInterfaceRingBreakDiagnosis the network cannot reach normal operation. In this state, a relative node position is determined in every device. This information can be used in case of a fatal error (ring break or defective device) to localize the error.

If there is no fatal error, the NetInterface immediately changes to state NetInterfaceNormalOperation.

In case of a Diagnosis Error Shut Down event, the position determined in each device describes the position relative to the device that was configured as Timing Master at the end of RingBreakDiagnosis (since there was no light at its input).

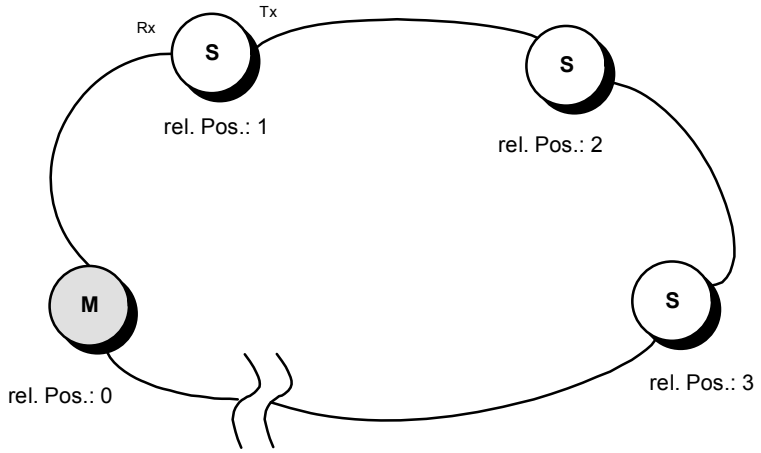


Figure 3-8: Localizing a fatal error with the help of ring break diagnosis.

Event	Transition to	Cause
Diagnosis Ready	NetInterfaceNormalOperation	No fatal error.
Diagnosis Error Shut Down	NetInterfacePowerOff	Fatal error (Ring break or defective device)

Table 3-9: Events in state NetInterfaceRingBreakDiagnosis

During RingBreakDiagnosis a device stays configured as Timing Master, until it recognizes light at its input, or until the Diagnosis Error Shut Down event is generated by occurrence of the timeout (t_{Diag_Master} or t_{Diag_Slave} respectively). On a fatal error, the application stores the error Error_Ring_Diagnosis with the relative ring position.

After recognition of a stable lock, a Timing Master device generates a Diagnosis Ready event and changes immediately to state NetInterfaceNormalOperation.

The timeout values (t_{Diag_Master} or t_{Diag_Slave}) can be changed by the system integrator, if alternative approaches for ring break diagnosis are used. In this case, the system integrator must make sure that all devices in the network are able to start the diagnosis process within the specified timeouts.

As soon as a device, which does not contain the Timing Master under normal operation conditions, recognizes light at its input, it is configured as Slave (bypass enabled). The bypass is deactivated after a recognized lock. If no lock errors occur for a time t_{Lock} (stable lock), the relative ring position is determined.

If, on stable lock, the bSBC register contains a value greater than 5, the ring is closed. There is no defect and the NetInterface changes to state NetInterfaceNormalOperation.

If the ring could be closed, every NetInterface switches to state NetInterfaceNormalOperation. The application will get notified about that by the NetOnEvent. After that, all high level initializations must be performed (Building of central registry, address initialization, notification, ...).

The following flow charts show the behavior in the state NetInterfaceRingBreakDiagnosis:

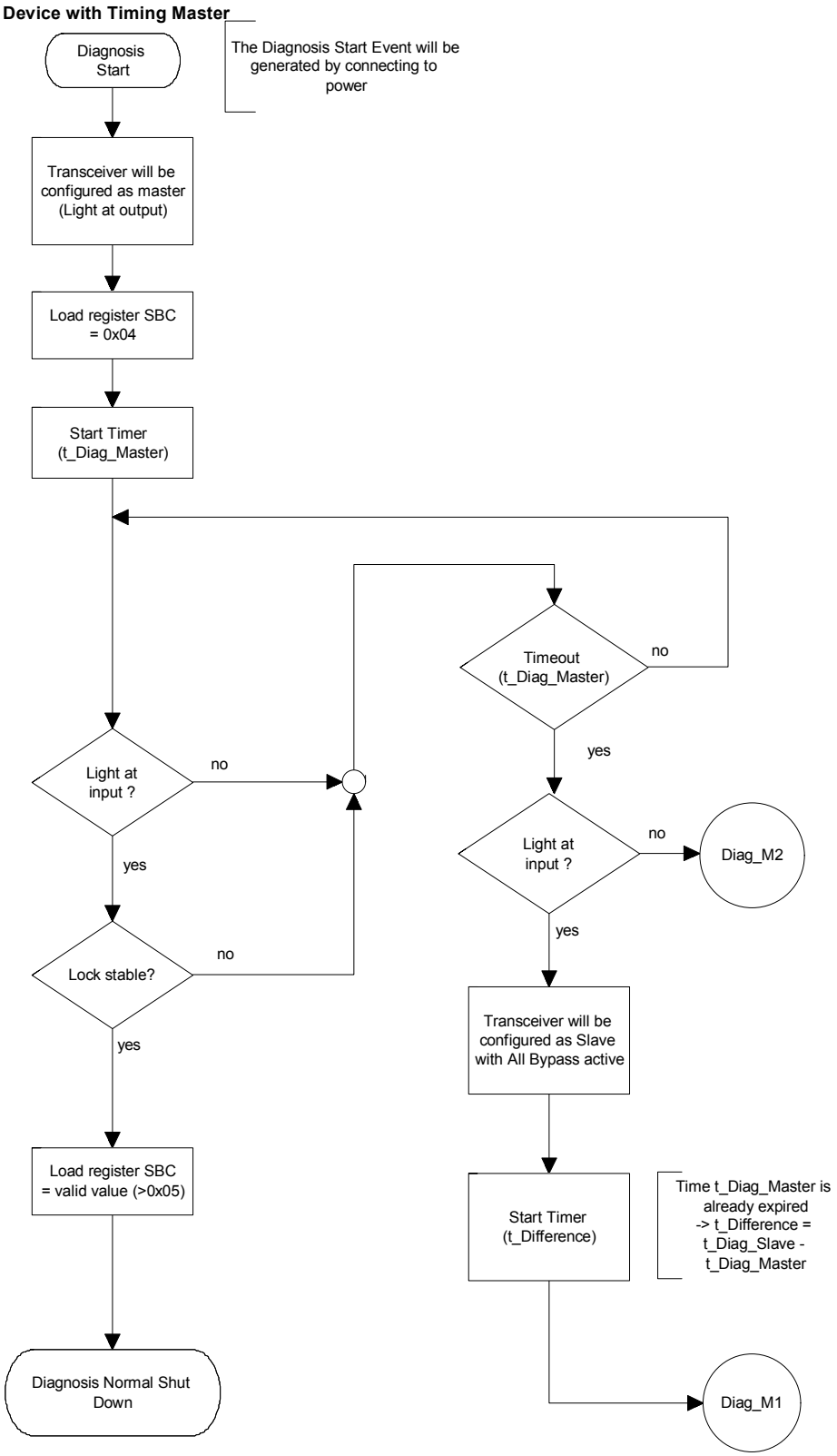


Figure 3-9: Behavior during ring break diagnosis in a Timing Master (part 1).

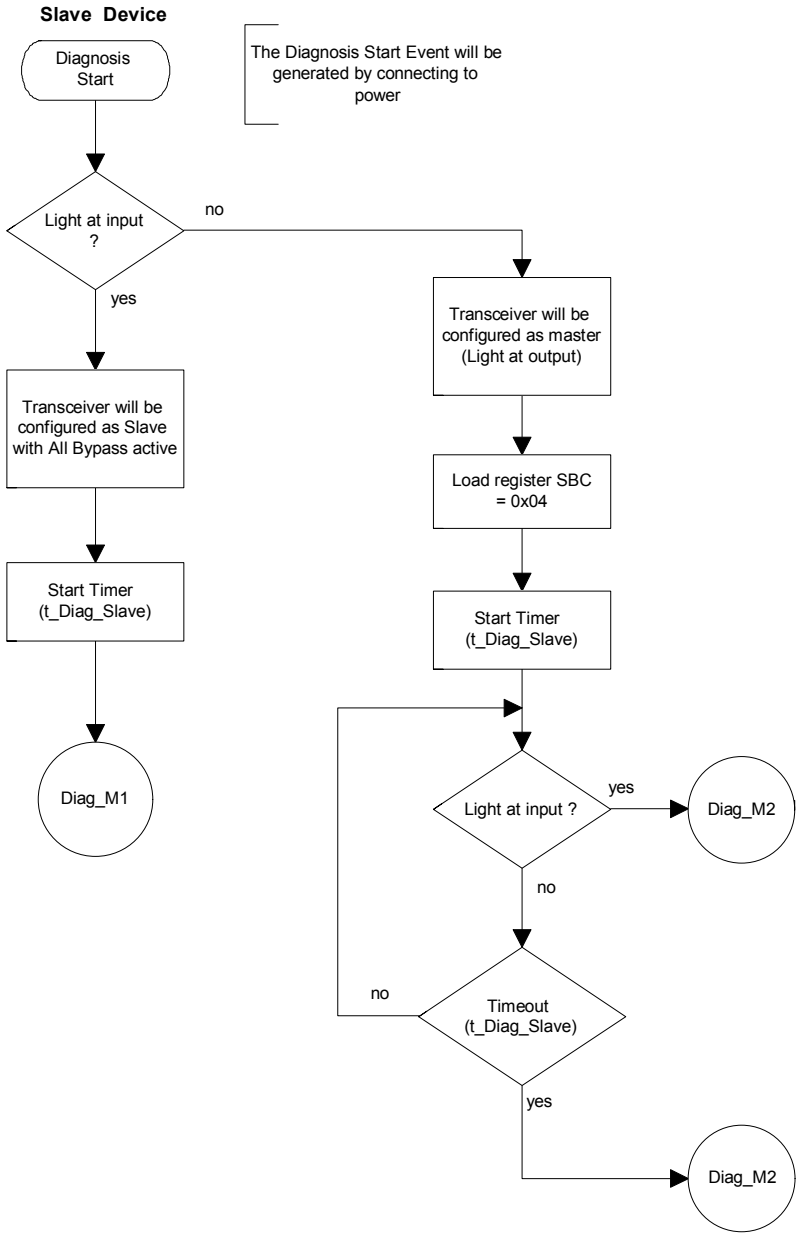


Figure 3-10: Behavior during ring break diagnosis in a Slave (part 1).

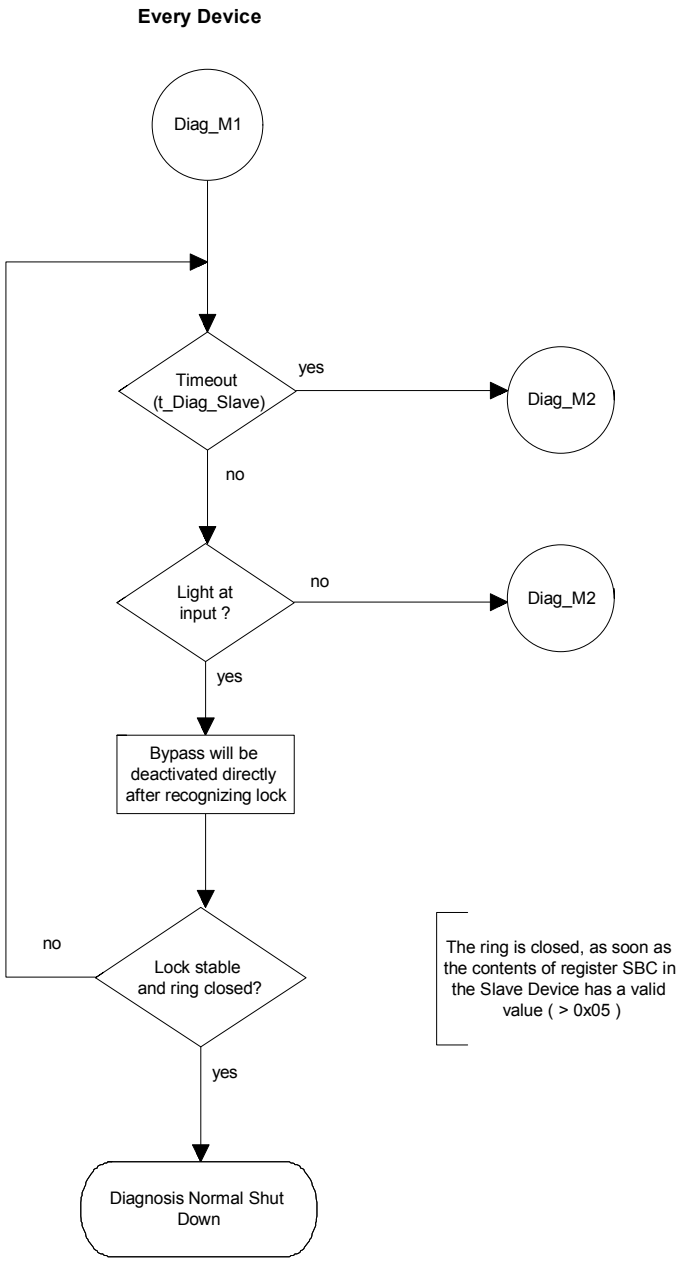


Figure 3-11: Behavior during ring break diagnosis in a Timing Master and Slave (part 2).

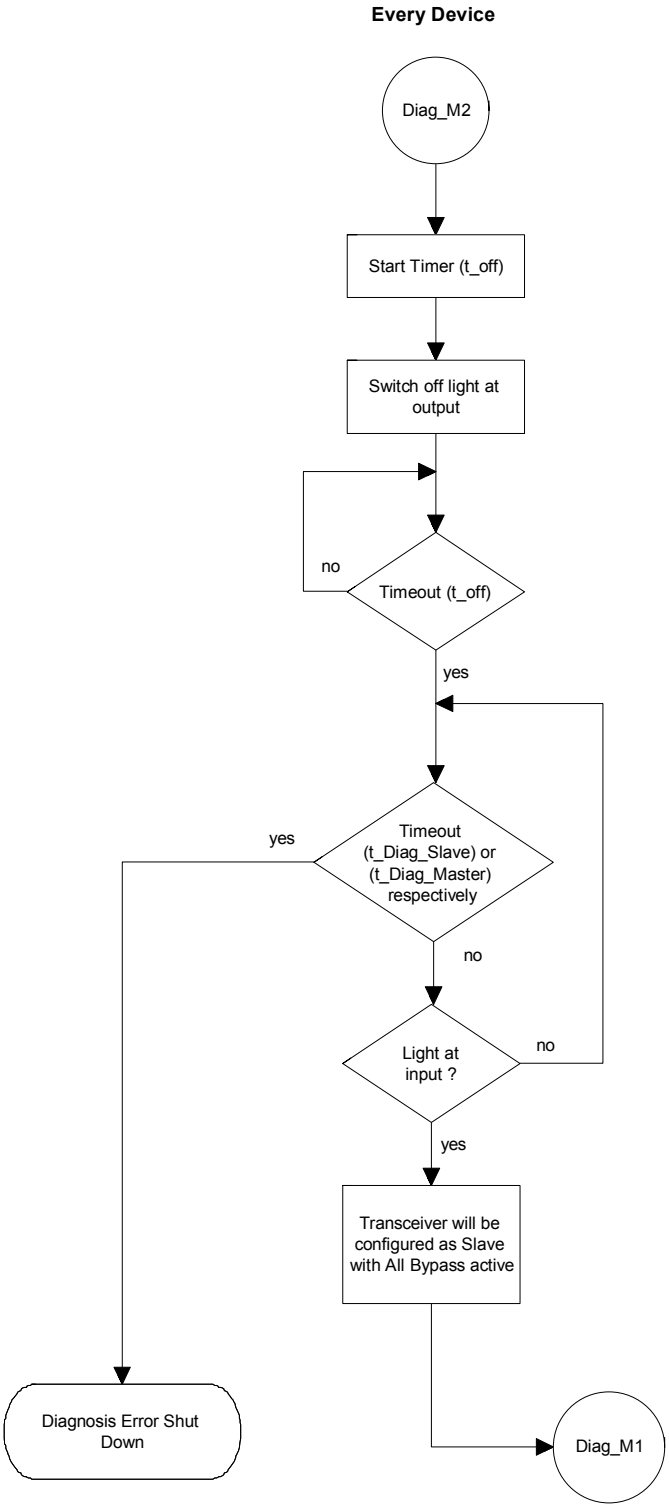


Figure 3-12: Behavior during ring break diagnosis in a Timing Master and Slave (part 3).

3.2.3 Initialization on Application Level

With the NetOn event, the NetInterface of a single device shows its local application readiness for communication. The network on a basic level is now ready for communication. Initializations can be performed that have to do with the interacting of multiple devices on the application layer, and the system (this does not mean that initialization of the individual applications shall start at this time and not earlier). The NetworkMaster checks system configuration. Depending on the result, the logical node addresses are initialized. Then the application can initialize the communication controlled by itself, i.e., it registers in the notification matrices. Initialization can be named "SystemCommunicationInit".

As a complicating factor, the system must be prepared for devices connecting to or disconnecting from the network (Network Change Event). In these cases, the system must run consistently without disturbances, and re-initializing phases must be as short as possible. On a NetworkChange event, parts of SystemCommunicationInit must be run again, but initialization must not be run completely due to the time this would take.

3.2.3.1 Requesting System Configuration – NetworkMaster

The NetworkMaster first checks to see if it has a logical node address stored in buffered RAM. If not, that means it was removed from power, and it assumes that the entire system must be initialized completely. It broadcasts Configuration.Status (NotOK), then determines its logical address based on its position.

If the NetworkMaster finds a valid node address (e.g. 0x0100) in its RAM, it writes it to the MOST Transceiver and starts checking the system configuration. (This is described in detail below.) If the logical addresses of the currently-available devices match the entries of the central registry from the last system run, and if no device has an un-initialized address (0xFFFF), the NetworkMaster broadcasts NetworkMaster Configuration.Status (OK). By doing this, the NetworkMaster tolerates devices that are missing now, but have been present in the old registry. This covers those cases, when devices register late. There are two kinds of "missing" devices:

1. **Devices having their All Bypass inactive (open).**
   It is mandatory, that every Device opens its All Bypass within time $t_{WaitNodes}$. By doing that, the device is available to the NetworkMaster during the request of system configuration. Although the application should be ready for communication within the same time as well, it can be that this does not happen in some devices. The NetworkMaster performs one complete request (Node 1 up to MaxNodes), ignoring nodes that do not respond. After that, he broadcasts the result as described above. The next round starts, after some time. This time, the NetworkMaster requests only those nodes, that did not answer during the first round. In case he retrieved some new information (i.e. one of the "silent" nodes has answered), he either broadcasts a supplementary registration:

   Configuration.Status(Control=New, FBlockIDList),

   as described in section 3.2.6.4, or

   Configuration.Status(Control=NotOK).

The NetworkMaster stops requesting only, in case all nodes have answered. This means, that after a system startup the NetworkMaster checks system configuration. The result can be either Configuration.Status(NotOK) (a new check will be performed), or Configuration.Status(OK) (indicates that the Central Registry has been re-built completely). In case additional nodes logically enter the system later, they will be integrated by a supplementary registration. The potential controllers for these nodes will be notified immediately by this mechanism. They therefore can integrate them in the system subsequently.

2. **Devices that have their All Bypass active (closed).**
   It can not be excluded finally, that a device keeps its All Bypass active during startup of the system (e.g. since it does not leave reset state). In case such a device enters the network late (Opening/ deactivating All Bypass), a NetworkChangeEvent is generated (for more information please refer to section 3.2.6.5). Triggered by the NetworkChangeEvent, the NetworkMaster checks the entire system configuration. In case the device has an initialized and valid address, only a supplementary registration of its Function Blocks is performed.

Only in case of a system shutdown in proper form (as a result of ShutDown.Start), the differences to the reference configuration are written to the error memory (Error_Registry_New). Writing is done only, when the system ran for at least $t_{Runtime}$ (starting at NetOn event). The same applies to the saving of the Central Registry in permanent memory (the system must have been running for at least $t_{Runtime}$, starting at NetOn event).

Example:

1. Starting situation:

Desired configuration:

Rx/TxLog	FBlockID	InstID
0x0101	AudioDiskPlayer	1
0x0102	AM/FMTuner AudioDiskPlayer	0 2
0x0103	TVTuner	0
0x0104	AudioAmplifier AudioAmplifier	1 2

central registry of last system run

Rx/TxLog	FBlockID	InstID	available?	Position
0x0101	AudioDiskPlayer	1		
0x0102	AM/FMTuner AudioDiskPlayer	0 2		
0x0103	TVTuner	0		
0x0104	AudioAmplifier AudioAmplifier	1 2		

2. System startup - Checking configuration: Devices 0x0102 and 0x0103 available

Rx/TxLog	FBlockID	InstID	available?	Position
0x0101	AudioDiskPlayer	1	no	
0x0102	AM/FMTuner AudioDiskPlayer	0 2	yes	1
0x0103	TVTuner	0	yes	2
0x0104	AudioAmplifier AudioAmplifier	1 2	no	

⇒ Broadcast Configuration.Status (OK)

3. Supplementary Registration: Device 0x0104 joins network (either physical through NetworkChangeEvent, or logical); Checking configuration

Rx/TxLog	FBlockID	InstID	available?	Position
0x0101	AudioDiskPlayer	1	no	
0x0102	AM/FMTuner AudioDiskPlayer	0 2	yes	1
0x0103	TVTuner	0	yes	2
0x0104	AudioAmplifier AudioAmplifier	1 2	yes	3

⇒ Broadcast Configuration.Status (Control=New, AudioAmplifier.1, AudioAmplifier.2)

4. ShutDown: Normal ShutDown with ShutDown.Start; Store error "Device 0x0101, AudioDiskPlayer.1 missed"

3b. Variant:
Un-initialized device 0xFFFF joins network at node position 2; Checking configuration; Recognizing un-initialized device in system; Broadcast Configuration.Status (NotOK); Building addresses; Building central registry

Rx/TxLog	FBlockID	InstID	available?	Position
0x0101	AudioDiskPlayer	1	no	
0x0102	AM/FMTuner AudioDiskPlayer	0 2	yes	1
0x0103	TVTuner	0	yes	3
0x0104	AudioAmplifier AudioAmplifier	1 2	yes	4
0x0102	Speech Recognition	0	yes	2

⇒ Broadcast Configuration.Status (OK)

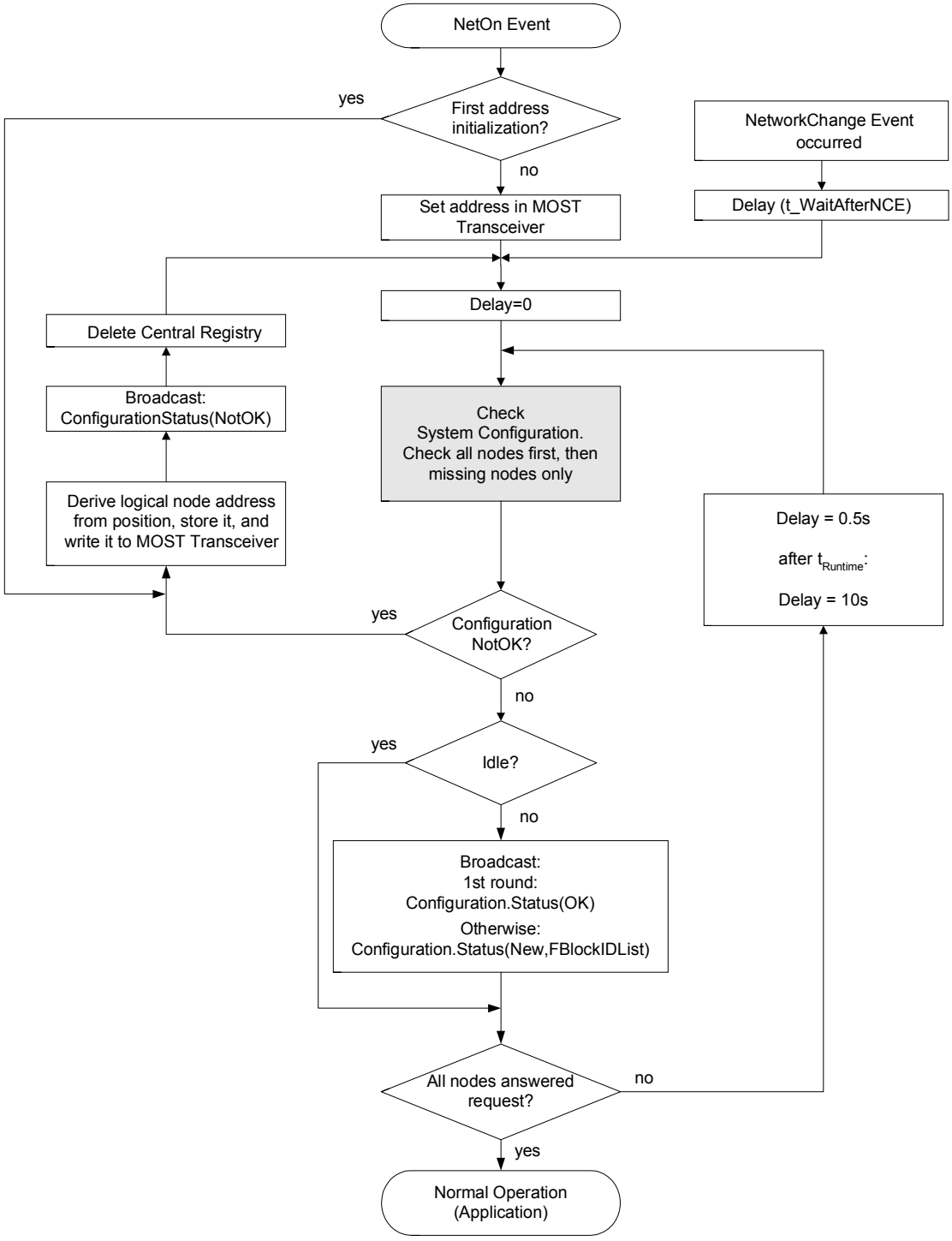


Figure 3-13: Flow of initialization on application level in a NetworkMaster.

For checking system configuration (Figure 3-14 on page 128), the NetworkMaster requests the FBlockIDs from each node in the network, using node position addressing. The logical node address is contained in the answer given by the addressed node, and is set in the MOST Transceiver. If the NetworkMaster recognizes an un-initialized device (Rx/TxLog=0xFFFF), the system configuration check is interrupted by Configuration NotOK. This also happens if two devices try to use the same address, or the assignment function blocks <-> logical address is no longer correct. If all devices are requested and their data corresponds with the entries in the stored central registry (logical address as well as contained function blocks), the process finishes with Configuration OK.

If one or more devices did not respond on the request of FBlockIDs, it is tried – after timeout - to request FBlockIDs again. The number of devices is derived from the MPR register of the MOST Transceiver. If a node caused result "NotOK" (for the system configuration check) the third time in direct succession, it will no longer be requested by the NetworkMaster until next system startup.

After the NetworkMaster has sent Configuration.Status(OK) first time during a system run, all applications, included in the device containing the NetworkMaster, are starting up. The same applies to the applications in the Network Slaves.

Please note:
The NetworkMaster starts supervising and storing of errors only for those devices, that have answered FBlockIDs.Get, and that are registered in the central registry.

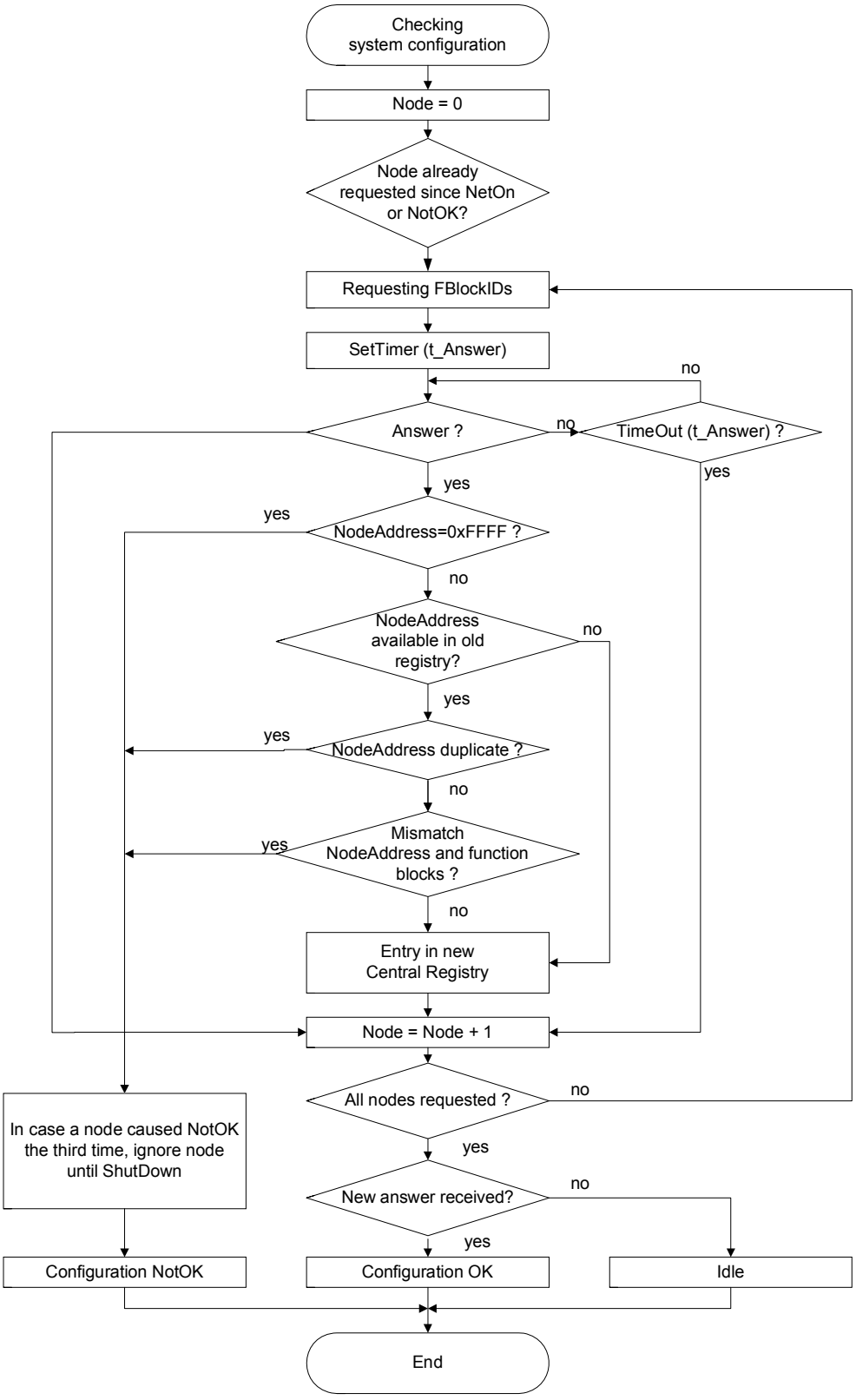


Figure 3-14: Flow in NetworkMaster during requesting system configuration.

The flow in Figure 3-14 on page 128 is shown in serial for clarification. The NetworkMaster should use a parallel approach, i.e. requesting several devices, and not waiting for answers before performing the next request.

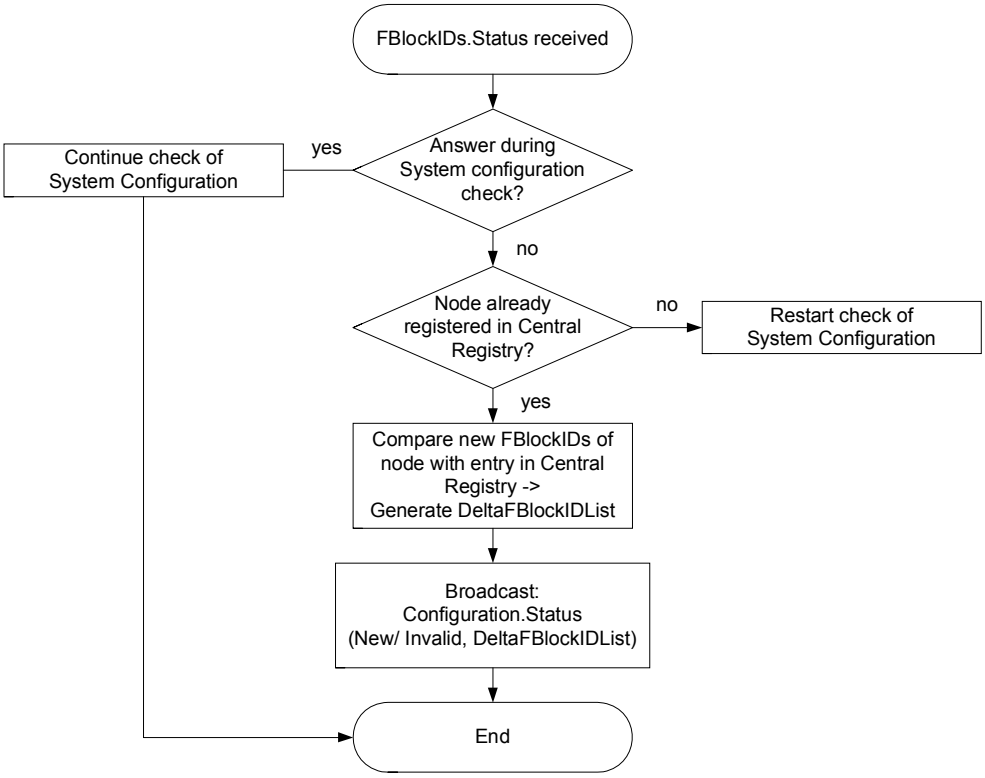


Figure 3-15: Behavior of NetworkMaster on receiving FBlockIDs.Status

3.2.3.2 Requesting System Configuration – Network Slave

All MOST Devices that do not contain the NetworkMaster are called Network Slave. The following diagram (Figure 3-16) contains the flow after the NetOn event in a Network Slave. At first, the logical address stored since last system run is restored, i.e. written to the MOST Transceiver. In case no logical address is available (e.g. since device was removed from power, or was powered first time), 0xFFFF is written to the MOST Transceiver. After having written 0xFFFF to the transceiver, the Slave waits for the Broadcast of Configuration.Status of the NetworkMaster. During its waiting, it answers the requests e.g. of FBlockIDs sent by the NetworkMaster.

In case either Configuration.Status(OK) or Configuration.Status(Control=New, FBlockIDList) is received, the Network Slave uses its current Decentral Registry. In case Configuration.Status(NotOK) is received, the Network Slave deletes its Decentral Registry and the logical node address of the last system run. It derives its new logical node address from its Node Position and writes it down to the MOST Transceiver. The Decentral Registry is re-built if required. In case of a regular ShutDown (i.e. triggered by ShutDown.Start), every device stores its Decentral Registry in buffered RAM. If the device is removed from power, it has to "forget" the Decentral Registry and its logical Node Address.

Please Note:
A Network Slave must not store the address of the NetworkMaster in non volatile RAM, or assume the address to be 0x0100. The address of the NetworkMaster must be derived during each system startup from the Broadcast of Configuration.Status.

After that, initialization of the application (with respect to communication) can start. During SystemCommunicationInit, e.g. Notification is established. Therefore, the application of a Device registers in the Notification Matrices of those Function Blocks, about which it desires to get status information.

It can happen, that the NetworkMaster performs a supplementary registration of Function Blocks through a new Configuration.Status. In case such a message is received, the routine described in Figure 3-16 must be activated. If such Function Blocks are registered the application is interested in, the application must either be initialized completely, or partially (e.g. registering in the Notification Matrices of the Function Blocks).

In case of a partial initialization caused by Configuration.Status(Control=New, FBlockIDList), Notification will not be deleted. A Controller simply has to register in the Notification Matrices of the new Function Blocks (in case they are relevant for the Controller's application).

Example:
A HMI controls AM/FMTuner, AudioDiskPlayer and NavigationSystem. First of all, only AM/FMTuner and AudioDiskPlayer available to the system. During the first run of SystemCommunicationInit, the HMI registers in the Notification Matrices of AM/FMTuner and AudioDiskPlayer. Caused by that, the current status information is returned to HMI. The control area (menus, buttons, displays etc.) of NavigationSystem are displayed "inactive" (e.g. in gray). The rest of the system does not require the functions of NavigationSystem, and therefore runs properly.

In case NavigationSystem enters the network, the NetworkMaster checks the system configuration and broadcasts Configuration.Status(Control=New, FBlockIDList), since the navigation system is already known, and its address is already initialized. For accelerating the following run of SystemCommunicationInit, the HMI should initialize only those sections that are relevant for NavigationSystem.

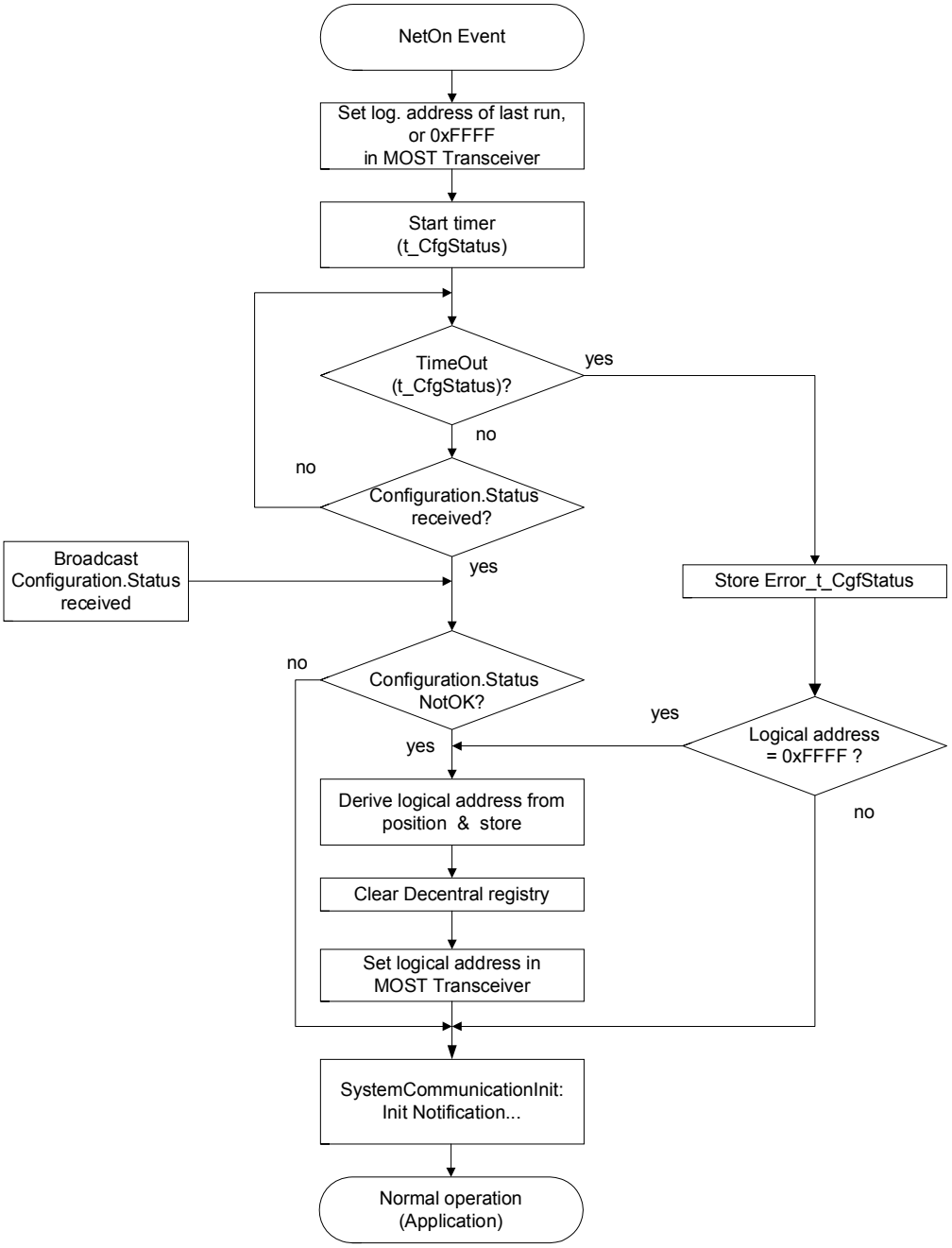


Figure 3-16: Flow of initialization on application level in a Network Slave.

3.2.4 Secondary Nodes

For some applications it can be useful to integrate two MOST Transceiver chips into one device. The two nodes are called "Primary" and "Secondary" node. A detailed definition of these names and a description of the possible structures are available in section 3.9 on page 181. A Secondary Node does not contain any Function Blocks (FBlocks). In case of receiving any request, it returns an error (Error Secondary Node. Please refer to section 2.3.2.5.1 on page 38). This error has the meaning of "I am a Secondary Node only. The node responsible for me has DeviceID 0xnnnn". The example below shows this mechanism by means of the request of System Configuration (NetworkMaster). During Network Configuration request, the NetworkMaster asks all nodes for the FBlocks they contain. The secondary node therefore replies:

```
SN -> NM: NetBlock.Pos.FBlockIDs.Error (ErrorCode = 0x0A = Secondary Node,
                                        ErrorInfo = DeviceID of Primary Node)
```

In the central registry it must be marked, which node is the Primary Node to a certain Secondary Node. Therefore, the Central Registry must be sorted in a way, that the Secondary Node's entry directly succeeds the entry of the Primary Node in case of a request. This does explicitly not refer to the hardware configuration in the respective device. In a MOST Device, the Primary Node can be arranged behind the Secondary Node as well.

For completing the entries of Secondary Nodes in the Central Registry, each Secondary Node is registered with a single Function Block (FBlock) having FBlockID.InstID = 0xFC.0.

3.2.5 Power Management

3.2.5.1 General Procedure

Power management means that the administrative function, which is above the NetServices, wakes and shuts down the MOST network. The power management is handled mainly by the function block PowerMaster.

- **Waking of the network:** Waking the network is done by emitting modulated light (light on). In principle the network can be awakened by any node. The ability of a node to wake up the network can be activated or deactivated by the PowerMaster (e.g. in case of a critical charge status of the accumulator) in the property **AbilityToWake**, which is implemented in every NetBlock. The PowerMaster itself will usually wake the network, for example, when there is communication on the car's bus, or based on the status of the vehicle (Clamp status).

  Please note:
  A device must only wake the network when this is initiated by the application. Failure (e.g., supply voltage too low, or too high) must not initiate waking of the network. Other solutions for waking up the network have been implemented as well, such as using an electrical wakeup line. It is up to the system integrator to choose the preferred wakeup method. The process described here, is independent of the wakeup method.

  When an application wakes the network, it calls the respective routine in the NetServices, which switches on light at the output of the device. Every node that recognizes light at its input switches on light at its output and initializes. In this way the light travels from node to node until the entire network is awake.

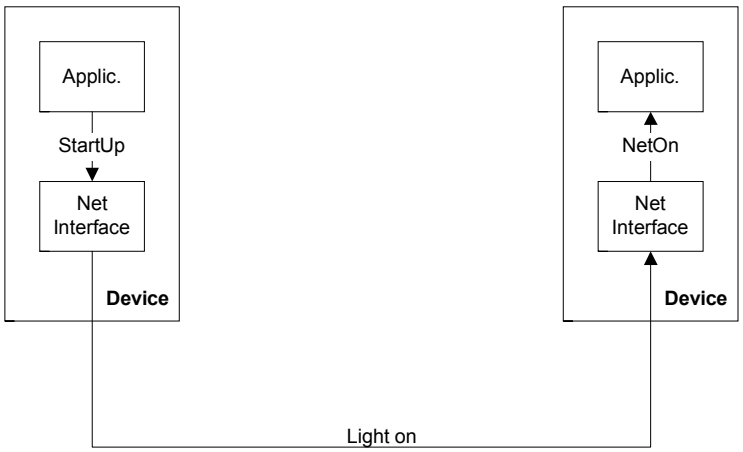


Figure 3-17: Example (2 devices) for waking of the MOST network via light on the network.

- **Switching off network:** Switching off the network is done on lowest level by switching off light. A device, which has switched off light, must not switch it on again before $t_{Restart}$ occurred. This applies also to that case, if it recognizes light at its input.

  All devices, except the one containing the PowerMaster, switch off light when certain errors occur (unlock, low voltage with reset). This is done without warning the other devices by sending a telegram.

In all other cases, only the PowerMaster switches off the network. For avoiding that devices have to save their status to non-volatile RAM very often, the PowerMaster implements a shutdown procedure that has two stages. This procedure contains request and execution. For requesting, it starts method **ShutDown** with parameter Query in all NetBlocks of the system. This is one of the rare cases where a telegram is broadcasted. After that, the PowerMaster waits for $t_{Suspend}$ before it actually shuts down the system. A device without any further need for communication does not respond on ShutDown.Start(Query). The execution is announced by the PowerMaster by starting ShutDown.Start(Execute).

By this function call, the shutdown process is started irrevocably. Every device has to prepare for shutting down without reply to the PowerMaster (Saving status) and waits for the light to be switched off.

The PowerMaster switches off light $t_{ShutDownWait}$ after ShutDown.Start(Execute). This time allows to shutdown audio output without audible side effects.

If a function block desires to communicate, it must notify the PowerMaster after ShutDown.Start(Query) with ShutDown.Result (Suspend) within time $t_{Suspend}$. The PowerMaster then postpones its attempt to switch off for time $t_{RetryShutDown}$, before retrying its shutdown. This procedure guarantees that a device which woke the bus in the parked vehicle does not need to prevent the PowerMaster from switching off the network actively (according to the current status of the vehicle). For switching off, the PowerMaster calls the respective routine in the NetServices. The status “light off” travels around the ring in the same way as “light on” when waking the network. After a certain delay time $t_{PwrSwitchOffDelay}$ the nodes change to sleep mode.

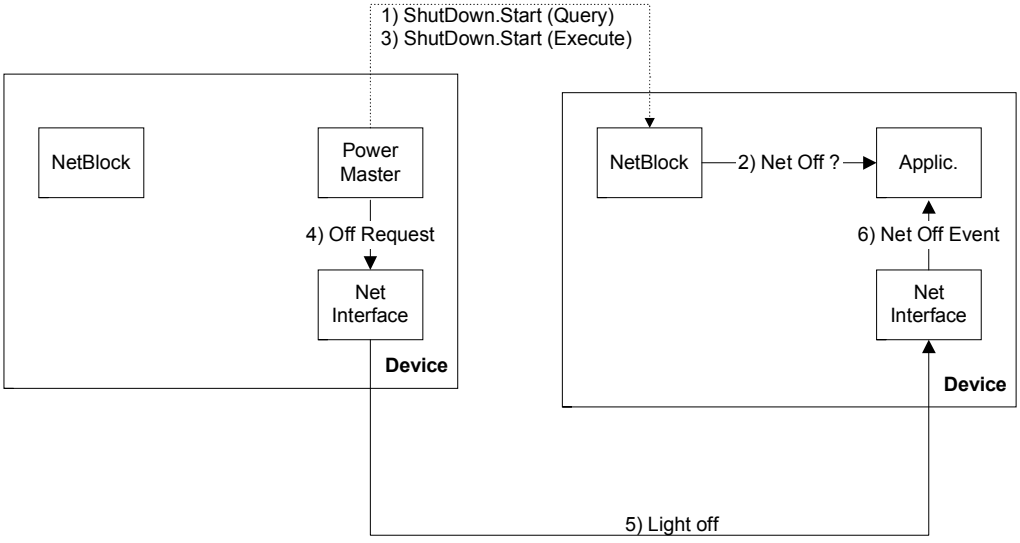


Figure 3-18: Switching off MOST network via starting method ShutDown in every NetBlock, and signaling to every application, and switching off light.

If a device desires to wake the network directly after a shutdown, it has to wait at minimum for $t_{Restart}$ (running from Light Off), before it switches on light again.

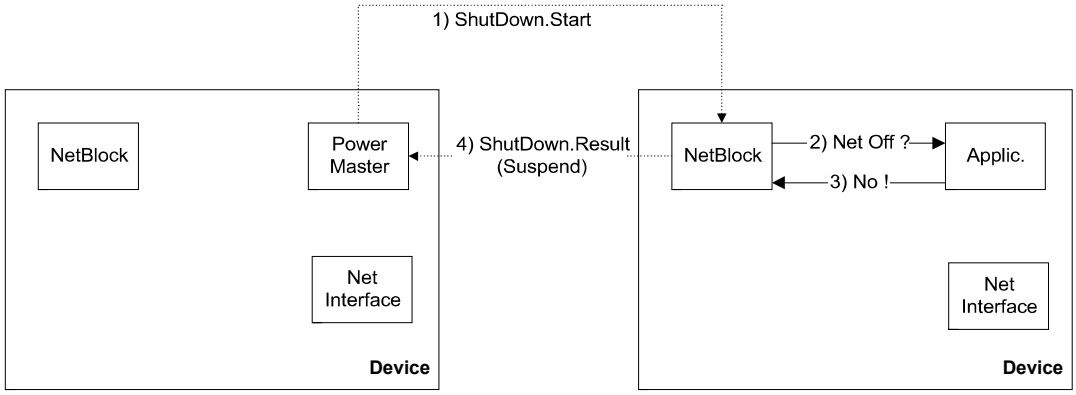


Figure 3-19: Prevention of switching off MOST network via ShutDown.Result (Suspend)

Please note:
If the light is switched off during startup, e.g. by low voltage, long unlock or fatal error, the PowerMaster must not wake the network for being able to finish its shutdown procedure. It will abort its startup procedure.

FUNCTIONS				
FktID	**OPType**	**Sender**	**Receiver**	**Explanation**
AbilityToWake	Set	PowerMaster	one NetBlock	On: Device can wake the network Off: Device must not wake the network
ShutDown	Start	PowerMaster	all NetBlocks	Announcing of switching off network
	Result	one NetBlock	PowerMaster	Reply on further demand for communication

Table 3-10: Functions in conjunction with power management.

3.2.5.2 Functions and Important Operations

The functions described above and the related operations are explained below with the help of the respective protocols.

AbilityToWake:

```
???  -> Slave: NetBlock.Pos.AbilityToWake.Set (WakeStatus)
```

WakeStatus	0x00	Off (Default for a non-waking device)
	0x01	On (Default for a waking device)
	0x02	Critical

```
???  -> Slave: NetBlock.Pos.AbilityToWake.Get
Slave -> ???:  NetBlock.Pos.AbilityToWake.Status (WakeStatus)
```

ShutDown:

Since the InstID is ignored in NetBlock, 0xFF must be regarded as a dummy value.

```
???  -> all: NetBlock.FF.ShutDown.Start (Query)

???  -> all: NetBlock.FF.ShutDown.Start (Execute)

Slave -> ???: NetBlock.Pos.ShutDown.Result (Suspend)
```

Query	0x00
Suspend	0x01
Execute	0x02

3.2.6 Error Management

In the network the following errors might occur on the lowest level:

- **Fatal Error:** Error that leads to the interruption of the ring, to the breakdown of the network, or that means the network can not be initialized (Super- or sub-voltage, ring break, defect FOT unit).
- **Unlock:** The PLL of the MOST Transceiver is no longer locked. A ring break is not necessarily the inevitable conclusion of this error.
- **Network Change Event:** One of the nodes in the network has activated or deactivated its bypass, which means it “disappears” or “appears” as a new node.
- **Voltage Low:** The voltage of one or more devices is too low to maintain operation of the NetInterface.

For the handling of these errors, there are the following general rules:

- **No alert communication:** For keeping error management simple, robust and not error-prone, there is no communication in case of an error.
- **Local Handling Of Errors:** Every device is responsible to handle every recognized error locally. Only the NetworkMaster handles errors for the entire network.
- **Securing Synchronous Signals:** In opposite to the packet data area and the control channel, there is no data securing for the synchronous area. So data transported here, is sensitive for disturbances in case of errors. So a device that recognizes an error, should secure all output signals in the synchronous area immediately. So a CD player has to mute the signal it feeds into the bus. The same applies to an audio amplifier, which has to mute its analog output signal (the one connected to the speakers). The synchronous connections on the Network are not removed, except in case of a fatal error.

3.2.6.1 Handling of Light Off

If a device recognizes at its input that light was switched off, it switches off its own output immediately. In case there is the need to wake the network again, it has to wait for $t_{Restart}$. If light was switched off without a ShutDown.Start (Execute), there might be two causes:

- Fatal error (voltage low, ring break), which is described below
- A device runs error handling (e.g. long unlock). In such a case the PowerMaster switches on light again, if the vehicle’s status requires it. So it wakes the network in the normal way. By that, a re-initialization is done.

If light was switched off, it might be the case that it is switched on again after a short time. If the application would shut down immediately, some devices might need a long time to return to normal operation. Therefore the application has to be prepared for ShutDown, but has to stay active for $t_{PwrSwitchOffDelay}$. If the light reappears within $t_{PwrSwitchOffDelay}$, the system is re-initialized like when waking up after sleep mode. The only difference is, that within the devices power supply, micro controller, and operating system need not to be re-initialized.

3.2.6.2 Fatal Error

A “fatal error” is a kind of error that prevents the light from being handed on in the Ring. There are four possible reasons:

- A device (especially optical transmitter, optical transceiver, or a MOST Transceiver) has no, or an insufficient distribution voltage.
- An optical receiver is defect.
- An optical transmitter is defect.
- The optical connection between transmitter and receiver is interrupted

3.2.6.2.1 Waking

If a fatal error occurs while an application tries to wake the network, “light on” does not propagate through the entire ring, and the NetServices in every device change to state NetInterfaceOff after t_{Slave}, or t_{Master}. The waking application waits for $t_{Restart}$ and then tries again to wake the network. This will be repeated up to three times, then it suspends the waking. Only the PowerMaster tries to start up the network if required by the vehicle’s status.

3.2.6.2.2 Operation

If there is a fatal error during normal operation, “light off” propagates through the entire ring. This is handled as described above. In case the power status of the vehicle requires it, the PowerMaster tries to wake the network after $t_{Restart}$. So the handling of a fatal error during waking needs to be performed (see above).

3.2.6.3 Unlock

An unlock occurs when a timing Slave cannot lock onto the input signal of the PLL of the MOST Transceiver, or if a Timing Master does not receive a comprehensible signal.

One cause for this might be that two Timing Masters in one ring work against each other. This case can be recognized only in the Timing Masters themselves.

Another cause can be that the optical signal at a node's input is too weak, or a node opens or closes its bypass. Every node downstream from the location that caused the unlock, up to the Timing Master, recognizes the unlock. The nodes downstream of the Timing Master up to the location that caused the unlock do not recognize the unlock. On an unlock, data errors occur. Based on its securing mechanism, the control channel is relatively insensitive to short unlocks.

Reaction of NetServices:
The NetServices of every device report an unlock immediately to the application by an Unlock event. In addition to that, the length of the unlock, and the occurrence of short unlocks is checked. If the network seems to be unstable due to long or frequent unlocks, an ErrorShutDown is performed by the NetServices. This is reported to the application with the help of a NetOff event. Following the context of its standard tasks, the PowerMaster tries to wake the network again after that.

Reaction on application level:
The application secures the synchronous signals. An audio amplifier must mute as fast as possible (refer to section 3.7.1.4 on page 178). After a lock is established again (recognized by the NetServices), the application restores its synchronous signals as fast as possible (e.g., de-mute). This must happen only if there is no NetworkChange event, where a node has closed its bypass and therefore has left the network.

Reaction on NetworkMaster:

The NetworkMaster does not react on an unlock in a specific way. Both errors are stored in the error memory in the same way than in other devices.

3.2.6.4 Failure Of A Function Block

It can be that single processes in a Device are hanging (but not the entire device), and that those processes need to be restarted. In case this failure stops an entire, or even several Function Blocks (FBlocks), the device has to un-register those FBlocks in the Central Registry in the NetworkMaster. This is done through a notification of the new status of FBlockIDs sent to the NetworkMaster:

```
Device -> NM: NetBlock.RxTxLog.FBlockIDs.Status (FBlockIDList)
```

This tells, that only those FBlocks contained in FBlockIDList are available. The NetworkMaster updates the Central Registry and broadcasts immediately after the reception of such an un-registration:

```
NM -> All: NetworkMaster.0.Configuration.Status   (Control=Invalid,
                                                   DeltaFBlockIDList)

Control              uns. Byte    0: NotOK
                                  1: OK
                                  2: Invalid
                                  3: New

DeltaFBlockIDList                 List of FBlockID.InstID
```

A detailed description of the handling of "Control = Invalid" and "Control = New" is to be found in section 3.2.3 on page 123.

DeltaFBlockIDList means the list of those FBlocks, that are invalid. Therefore, all application have the required information and can terminate functions depending on the invalid FBlocks.

If the failed process is ready (after being killed and re-initialized), the depending FBlocks are registered again:

```
Device -> NM: NetBlock.RxTxLog.FBlockIDs.Status (FBlockIDList)
```

The NetworkMaster registers those FBlocks in the Central Registry, and broadcasts immediately after having received the registration:

```
NM -> All: NetworkMaster.0.Configuration.Status (Control=New,
DeltaFBlockIDList)
```

Here, DeltaFBlockIDList means the list of the new FBlocks.

Please note:
In case a device, that starts up fast, has single FBlocks starting up relatively slow, the same mechanism of supplementary registration can be used.

The NetworkMaster handles 5 FBlocks max. (supplementary registration, un-registration). Therefore, the message can be sent within a single telegram (12 bytes max.). This refers to such nodes, which can handle single telegrams only. In case more than 5 FBlocks must be handled, several single telegrams are sent.

3.2.6.5 NetworkChange Event

A NetworkChangeEvent occurs, if a device opens or closes its All Bypass, i.e. enters or leaves the network. A NetworkChangeEvent is recognized by the NetServices in every device, by a change of the MPR register in the MOST Transceiver.

If a short unlock occurs during the NetworkChangeEvent, it is caught by the corresponding error handling. If an additional node joined the network, the new node must be integrated on system level. Therefore SystemCommunicationInit must run, but only partially. In addition to that, the NetworkMaster checks configuration again, and a jump to initialization with Configuration.Status is performed in each device.

If a node has left the network, the output signals that depend on synchronous data transfer must be secured immediately. They must not be restored after the disappearance of the (eventually short) unlock, since it might be that the source of a synchronous signal is missing. Furthermore, every node must be able to handle the case where a communication partner is missing, and must terminate dependent parts of the application in a safe way. Also in this case the NetworkMaster checks system configuration, but without broadcasting Configuration.Status.

3.2.6.6 Low Voltage

For exact values, hysteresis etc., refer to section 4.7 on page 194 please. A too-low supply voltage does not inevitably occur in every device at the same time and in the same intensity. As already described, there are two limits regarding the supply voltage of a device:

Critical voltage $U_{Critical}$:
First, there is the limit at which the application will no longer work safely, but where communication is still possible. Since the application does not work any longer, the output signals that depend on synchronous data transfer must be secured. In case of a recovery, they can be restored immediately.

Low voltage U_{Low} :
There is a second limit, where even the NetInterface no longer works reliably, so even communication cannot be maintained. For being able to survive short intervals of low voltage, a Power Save Mode should be implemented:

> **Power Save Mode:** During the initialization process, the initializing of operation systems, initialization of the application, and the communication takes most of the time, while the actual network is initialized within approximately 100ms. At low voltage the re-initialization of operating systems, the application, and communication must be prevented for as long as possible. Especially in case of devices that have long initialization intervals. Suitable actions to be taken are: buffering of supply voltage, unloading protection, switching off peripherals of the application, resetting of the MOST Transceiver (closes its all bypass), stopping the controller, etc. For avoiding to disturb communication on the network, light must not be switched off in Power Save Mode. The device containing the Timing Master, must not close its All Bypass.

If the low voltage cannot be “survived” and the device is reset, then it switches off the light, rejects the initialization states, and switches to normal DevicePowerOff Mode, as if it was switched off. The device stays in DevicePowerOff mode, even if the supply voltage recovers. It is awakened either by “light on” at its input, or by the demand for communication from its own application. It changes to mode DeviceNormalOperation via the standard initialization process. The low voltage reset leads a device to normal behavior.

By opening the bypass, the device indicates that it is joining the network. The other devices must then integrate it into the system via SystemCommunicationInit. Further signaling is not required here as well.

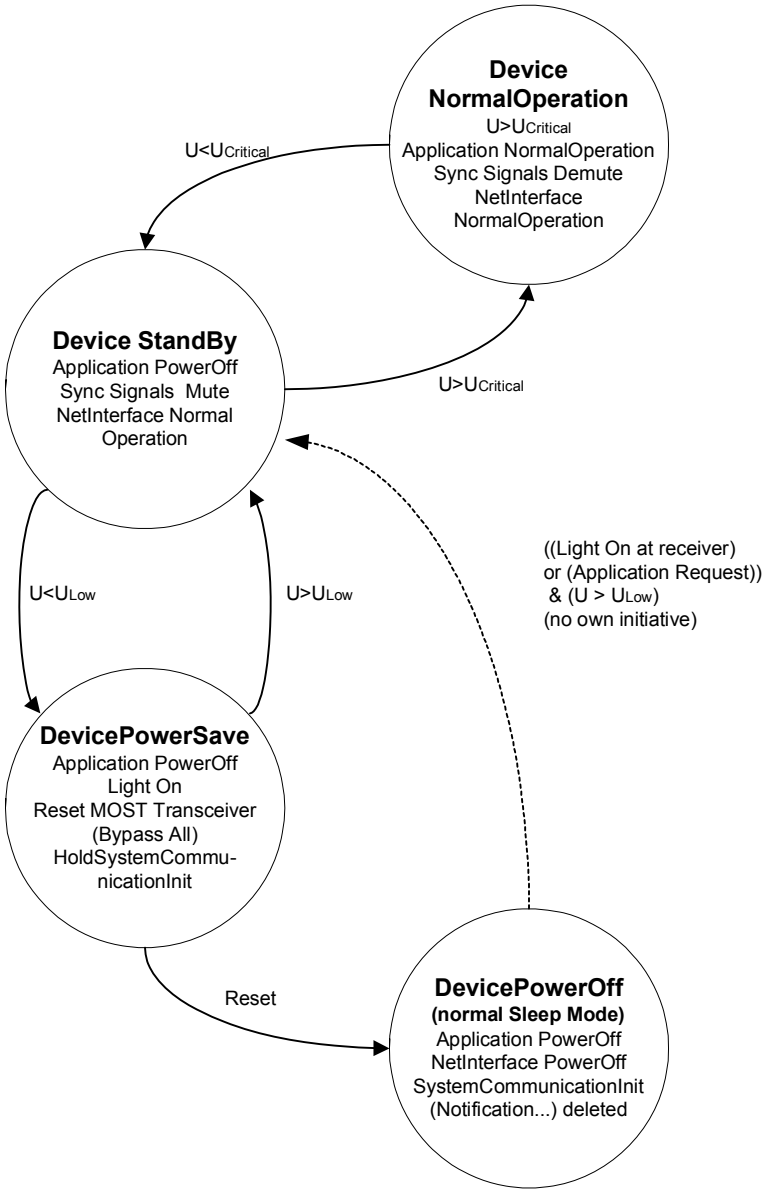


Figure 3-20: Behavior of a device depending on supply voltage.

3.2.6.7 "Hanging" of an Application

By implementing a watchdog in each device, a long "hanging" of the application should be avoided. This effect is reduced to a NetworkChangeEvent (Closing of All Bypass), and an eventual second NetworkChangeEvent (Opening of All Bypass).

Every application must be able to handle the case where if one of its communication partners does not respond, and safely terminate the parts of the program that depend on this communication.

3.3 Accessing Control Channel

3.3.1 Addressing

In a MOST network, nodes in a ring are addressed. The MOST Transceiver provides four different types of addresses, which are described below.

- **Node Position Address (*Rx/TxPos*)**
  Physical position of the transceiver in the ring (0x00 up to 0xFF). Counting for this address starts at 0x00 in the Timing Master node. It is called RxPos for a receiving node, and TxPos for a transmitting node.
- **Logical Node Address (*Rx/TxLog*)**
  User definable address between 0x0001 up to 0x02FF, and 0x0500 up to 0xFFFF. It must be unique in the system, and is called RxLog for receiving nodes and TxLog for transmitting nodes.
- **Group address**
  Provides access to a group of devices. Valid addresses are 0x00 up to 0xC7, and 0xC9 up to 0xFF.
- **Broadcast address (0x03C8)**
  All devices.

Addressing is done in the following way:

Node Position Address:

A node position address is unique by definition, but it has the disadvantage that the receiving MOST Transceiver does not report the sender's node position address, but the sender's logical node address. This happens only when using node position addressing. For this reason, node position addressing is not used under normal operation conditions. It is used by the NetworkMaster only for administrative tasks, such as during initialization. A node position address can be determined using the **NodePositionAddress** function in the NetBlock. It consists of an offset plus the position value:

Rx/TxPos = 0x0400 + Pos

Pos = 0 for Timing Master
Pos = 1 for first device in ring...

Logical Node Address:

Logical node addressing is used by all nodes to address a single node. Only the NetworkMaster uses the node position address (administration only). The section below describes the default procedure for assigning logical node addresses.

A logical node address must be unique even if there are multiple devices of the same type. Therefore, it is derived from the unique node position address. During initialization of the network, the logical node address is calculated by each device as follows:

Rx/TxLog = 0x0100 + Pos

So the system Master device (containing the Timing Master) at position 0x00, will have the logical node address 0x0100, and a device at position 5 in the ring will have address 0x0105.

Another approach is to assign certain address ranges with respect to the functionality of devices. That means, for example, that the first video display module in a network gets address 0x0200, the second 0x0201, etc., while the first active amplifier gets address 0x0188.

The logical node address can be requested from the function **NodeAddress** in the NetBlock.

For checking whether a logical node address is unique, the MOST Transceiver provides a special procedure (SAI) which is described in section 3.1.5.2 on page 105.

The logical node address is stored in unbuffered RAM, so it is lost if the device loses power for some time. If the device stays powered, the logical node address is kept. After first power up, the logical node address is set to 0xFFFF (refer to section 3.2.3 on page 123).

Group Address:

The group address can be requested from function **GroupAddress** in the NetBlock, it can be modified using this function if required. The default procedure for deriving a group address is to take the FBlockID of the function block that is most characteristic for the device:

GroupAddress = 0x0300 + FBlockID

The function block (FBlockID) that is reported first in case of a request for the FBlockIDs is typically the most descriptive for the device.

Groups can be built dynamically by modifying group addresses.

The group address is stored in unbuffered RAM, so it is lost if the device loses power for some time. If the device stays powered, the group address is kept. In case power gets lost, the default value (FBlockID) must be restored.

Broadcast Address:

Broadcast addressing requires a great deal of system resources and therefore should be used for administrative tasks only.

FUNCTIONS				
FktID	**OPType**	**Sender**	**Receiver**	**Explanation**
NodePositionAddress	Get	Controller	NetBlock	Requesting Node Position Address
	Status	NetBlock	Controller	Answer
NodeAddress	Get	Controller	NetBlock	Requesting Logical Node Address
	Status	NetBlock	Controller	Answer
GroupAddress	Get	Controller	NetBlock	Requesting Group Address
	Status	NetBlock	Controller	Answer
	Set	Controller	NetBlock	Setting Group Address

Table 3-11: Functions in NetBlock that handle addresses

3.3.2 Assigning Priority Levels

Despite the high capacity of the control channel, temporary overload situations are possible, for example, during system initialization. Nevertheless, it must be possible to send important messages in that case. To do this, a fair arbitration mechanism is implemented in the MOST Transceiver. Four priorities are defined that are assigned to different function blocks via the mXCMB Register (Default = 0x01; 0x00 = lowest Priority):

Priority	Used for function blocks that
0x00	are controlled
0x04	have administrative tasks
0x08	handle control events initiated by the end user
0x0C	reserved

Table 3-12: Priorities on the control channel

3.3.3 Low Level Retries

In case the sending of a control message is not successful, MOST Transceiver can re-send the message automatically. Registers specify the number of retries and the delay between the retries. Typically, these values should not be changed, however they can be modified to fine tune the system.

3.3.4 High Level Retries

High level retries are not planned at this time, since the expenditure in software development would be too great for the expected results. All devices have to safeguard the ability to accept messages within the interval of time given by the low level retries.

3.3.5 MOST NetServices (Application Socket)

3.3.5.1 Basics for Automatic Adding of Physical Address

Since applications know only functional but not physical addresses, a protocol that is transported must be complemented by the physical address (DeviceID). There are two possible ways to achieve this.

One way is when the application answers a request. In this case it already has the DeviceID of the receiving node because it was reported during the request. The other way is when the application is sending a protocol and does not know the DeviceID of the receiver. In this case it sets the DeviceID to 0xFFFF. The ID is complemented by the NetServices and inserted into the MOST telegram as RxAdr.

3.3.5.2 De-Central Registry

The complementing of the DeviceID assumes that the assignment between a functional address (consisting of FBlockID.InstID and the logical node address) is known by the NetServices. If required, the NetServices build a De-central Registry, which contains the respective assignments:

Functional address (FBlockID.InstID)	Device containing the FBlock (logical node address = DeviceID)
AudioAmplifier.1	0x0105
AudioAmplifier.2	0x0103
AM/FMTuner.0	0x0107
AudioDiskPlayer.1	0x0107

Table 3-13: Example for a De-central Registry.

Only devices that control other devices need a De-central Registry. Devices that are controlled only, that is, those that receive requests and answer only those requests, for example, drives or tuners, get the DeviceID of the requesting node along with the request, and so do not need a De-central Registry.

To build a De-central Registry, i.e. for seeking a function block in the network, there are different possibilities. If there is a Central Registry, a request should be sent there, since this approach saves resources. See the next section for more explanation.

3.3.5.3 Central Registry

In larger networks it might be useful to build a Central Registry. The NetworkMaster generates it during initialization of the network. It is kept in buffered RAM (i.e. it is deleted if the device is removed from power) and contains the logical node address for each node, and the respective function blocks:

Rx/TxLog	Rx/TxPos	FBlockID	InstID
0x0100	0	AudioDiskPlayer	1
		NetworkMaster	0
		ConnectionMaster	0
0x0101	1	AudioDiskPlayer	2
0x0102	2	AM/FMTuner	0
		AudioTapeRecorder	0
0x0103	3	AudioAmplifier	2
etc.			
MaxNode	MaxNode	ManMachineInterface	0

Table 3-14: Central Registry

The Central Registry checks the system configuration, and finds communication partners, or their physical addresses.

The Central Registry is saved on SystemShutDown. On a restart of the system, the NetworkMaster asks all nodes for the information shown above. It then receives the logical node address automatically.

```
NetworkMaster -> Device: NetBlock.Pos.FBlockIDs.Get

Device -> NetworkMaster: NetBlock.Pos.FBlockIDs.Status (FBlockID.InstID,
                                                        FBlockID.InstID...)
```

If there is a contradiction between an entry in the Central Registry and the received data, the NetworkMaster rejects the stored Central Registry and broadcasts:

```
NetworkMaster -> all: NetworkMaster.0.Configuration.Status (NotOK)
```

This forces all devices to reject their De-central Registries, and to re-initialize their logical node addresses (based on node position). The De-central Registries are re-built on demand, i.e., not directly after Configuration.Status (NotOK). This means an entry of the De-central Registry is built if the application in a device desires to send a protocol without knowing the physical address of the receiving node.

If there is no contradiction between entries in the Central Registry and the received data after all nodes have been requested, the NetworkMaster broadcasts:

```
NetworkMaster -> all: NetworkMaster.0.Configuration.Status (OK)
```

Based on that, all other devices restore their De-central Registry based on the stored copy.

FUNCTIONS				
FktID	**OPType**	**Sender**	**Receiver**	**Explanation**
FBlockIDs	Get	Network Master	NetBlock Slave	Request of all function blocks contained in a device
	Status	NetBlock Slave	NetBlock NetworkMaster	Answer
Configuration	Status	Network Master	NetServices of all Slaves	Event, by which the NetworkMaster tells all slaves whether the system configuration is identical to the one of the last system run

Table 3-15: Commands in conjunction with the Central Registry.

If available, the Central Registry must be used if the NetServices of a device seek a physical address. With the help of the telegram Configuration.Status, the slaves recognize the logical address (Rx/TxLog) of the NetworkMaster containing the Central Registry. They ask the NetworkMaster for the physical address of a desired function block. The NetworkMaster takes the respective logical node address (Rx/TxLog) from the Central Registry and sends it to the Slave. The slave's NetServices ask:

```
NetworkMaster.0.CentralRegistry.Get (FBlockID.InstID)
```

The NetworkMaster answers:

```
NetworkMaster.0.CentralRegistry.Status (Rx/TxLog.FBlockID.InstID)
```

If there is no matching entry in the Central Registry, the NetworkMaster answers:

```
NetworkMaster.0.CentralRegistry.Error (ErrorCode, ErrorInfo)
```

with ErrorCode = 0x07 (Parameter not available) and returning of the parameter number (1 in this example), and the parameter which is not available (FBlockID.(InstID) in this example) as ErrorInfo.

It is possible to request the entire Central Registry:

```
NetworkMaster.0.CentralRegistry.Get ()
```

The NetworkMaster replies with all entries of the Central Registry:

```
NetworkMaster.0.CentralRegistry.Status (Rx/TxLog.FBlockID.InstID,
                                        Rx/TxLog.FBlockID.InstID...)
```

FUNCTIONS				
FktID	**OPType**	**Sender**	**Receiver**	**Explanation**
CentralRegistry	Get	NetServices Slave	NetworkMaster	Requesting a physical address
	Status	Network Master	NetServices of requesting Slave	

Table 3-16: Functions for building a De-central Registry if a Central Registry is available.

Please note:
When seeking the logical address of a communication partner, a device performs the following flow:

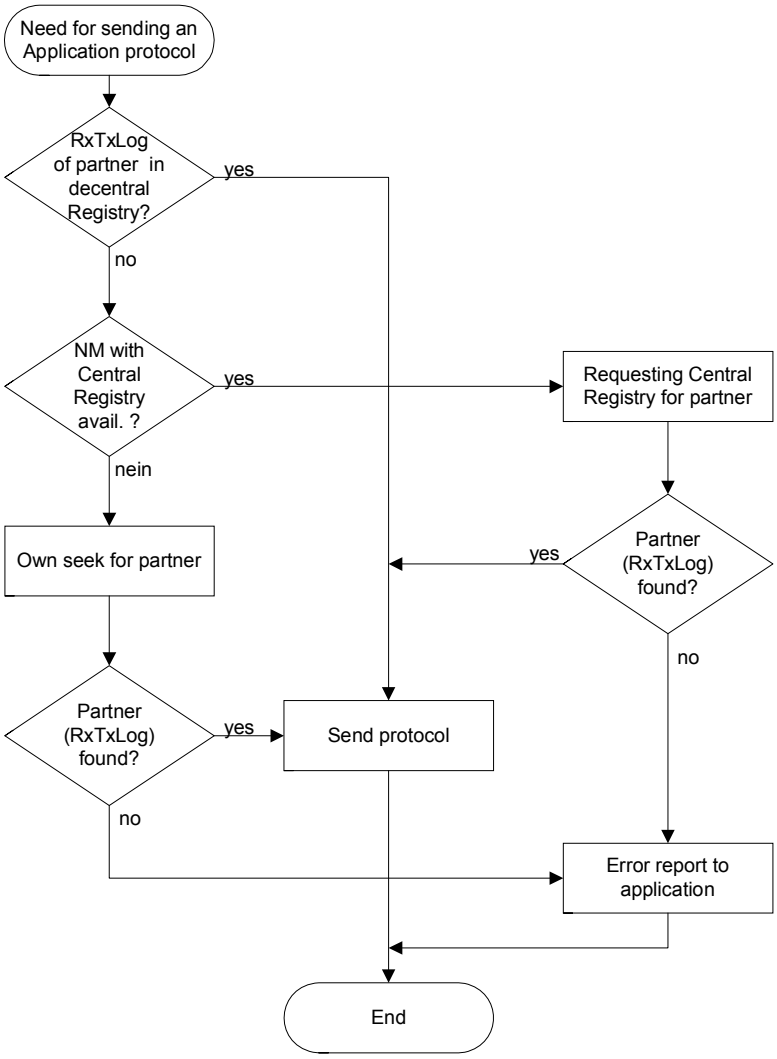


Figure 3-21: Seeking the logical address of a communication partner

3.3.6 Handling Overload in a Message Sink

The MOST Transceiver informs the sender's NetServices by a NAK error message indicating that the receiving node has rejected a telegram although the low level retries were used. This is an indicator for a momentary overload, or a defect. The NetServices pass the NAK error message through to the application, which has to decide what needs to be done (retry, reject, postpone). The error is stored as Error_NAK.

If that telegram belongs to a connection where data is sent continuously from a sender to a receiver, an optional mechanism can be implemented, which adapts the telegram transfer rate to the speed of the data sink. A simple mechanism might look like this:

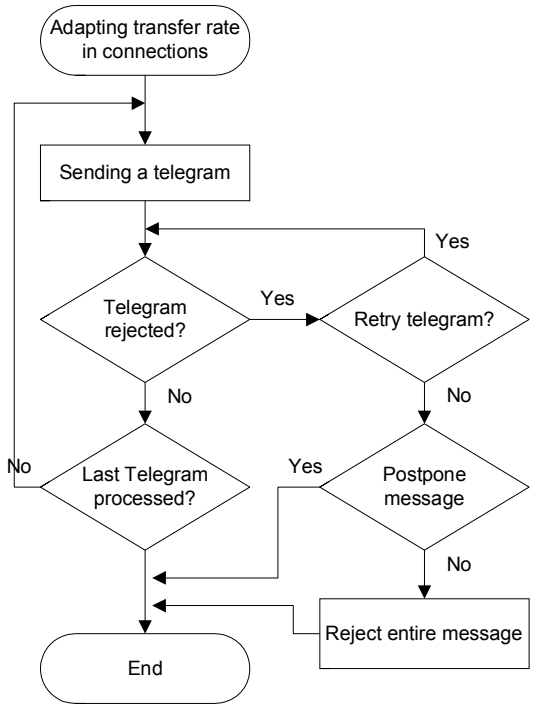


Figure 3-22: Possible mechanism to adapt transfer rates to the speed of a data sink.

It is assumed here, that errors due to incorrect address, or CRC error are handled "on top" of that mechanism. "Message" refers to the entire amount of data to be sent. A telegram is that portion of data, which can be transported on the Control Channel. It transports a part of the entire message. Rejecting a telegram means, that the target node could not process it due to an occupied receive buffer. In that case, the MOST Transceiver has already run its low level retries. Now the application has three selections:

1.) The telegram can be sent again, thus having additional low level retries available.

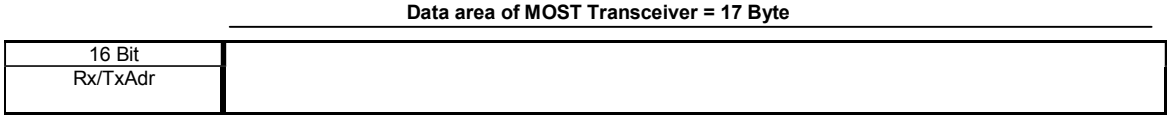
2.) The entire message can be rejected, e.g. because it is no longer relevant.

3.) The entire message can be postponed, i.e. sent later.

3.3.7 MOST NetServices (Basic Layer)

3.3.7.1 Control Message Service

Via the MOST Transceiver, MOST telegrams can be sent and received which consist of a sender or receiver address respectively (Rx/TxAdr), and a maximum number of 17 data bytes.



The lowest layer of the NetServices provide a mechanism which is called control message service (CMS). It handles the setting and reading of the registers of the MOST Transceiver.

3.3.7.2 Application Message Service (AMS) And Application Protocols

MOST NetServices provide three different types of transmissions via the control channel. Two of them are mandatory for each device:

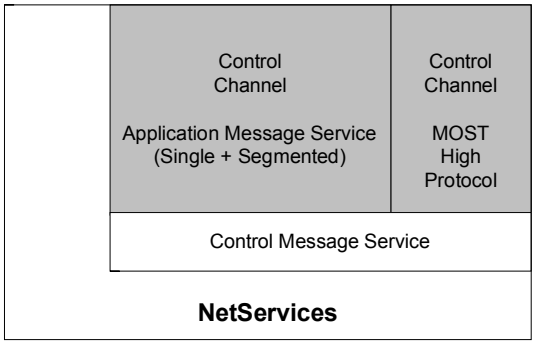


Figure 3-23: NetServices: Services for control channel

- **Single Transfer:** Data packets up to twelve bytes are transmitted in a single telegram.
- **Segmented Transfer:** Commands and status messages with a length greater than 12 bytes are transported by multiple telegrams.

Single as well as segmented transfers are based on the application message service (AMS), which is mandatory for all NetServices. In addition to that, a third transmission procedure is defined:

- **MOST High Protocol:** For connections, that is, the transmission of data streams or the transmission of larger data packets, a higher transport protocol, the MOST High Protocol can be used, which is derived from the well-known Transport Control Protocol (TCP). It uses some of the mechanisms defined by TCP, but can only be used for communication within the MOST network. MOST High Protocol transports data and could be used for transporting data coming from the external world (GSM) that is secured by the "real" TCP. For more detailed information about MOST High Protocol, please refer to [3].

As already described in section 2.3.2 on page 32, protocols of the following type must be transmitted:

```
DeviceID.FBlockID.InstID.FktID.OPType.Length (Parameter)
```

The application message service (AMS), is based on the control message service (CMS). MOST telegrams transport application protocols. Each telegram is divided up as follows:

Data area of MOST Transceiver = 17 Byte

16 Bit	8 Bit	8 Bit	12 Bit	4 Bit	4 Bit	4 Bit	8 Bit	8 Bit		8 Bit
DeviceID	FBlock ID	Inst. ID	Fkt ID	OP Type	Tel ID	Tel Len	Data 0	Data 1	...	Data 11

The parts of the application protocol are cross-hatched. The length of the application protocol is not transmitted directly. It has no meaning on telegram level, since several telegrams might be required to transport one protocol. Nevertheless, length is transmitted indirectly via TelLen and MsgCnt and must be restored on the receiver's side.

TelID: Identification of kind of telegram

Meaning	TelID	Data 0 = MsgCnt
Single Transfer	0	Data 0
1st telegram Segmented Transfer	1	0x00
2nd telegram Segmented Transfer	2	0x01
..	2	..
..	2	0xFF
..	2	0x00
..	2	..
(n-1). Telegram Segmented Transfer	2	0x(n-1)
Last telegram Segmented Transfer	3	0xn
MOST High Protocol User data	8	
MOST High Protocol Control data	9	

TelLen: = 0...12 specifies the length of the data field, i.e. ,the number of bytes after TelLen; 0 means no data byte

Data 0-Data 11: Data bytes

3.3.8 Direct Access to OS8104

3.3.8.1 Sending Messages

In order to transmit a message, the “Message Transmit Section” must be loaded by an external micro controller. The address of the receiving node (or the group of nodes) that should receive the message, the message type, the priority, and the message itself must be written here. The message type specifies whether the message is a standard control command, a remote command, or a resource command. Setting the retry values should be done by external network management, and should not be changed for each message separately.

After loading the message, only the “Start of Transmission Bit (STX)” in the message control register (bMSGC) must be set. After transmission the chip generates an interrupt (only if enabled), which indicates that the status of transmission (successful or not) is available in the respective registers.

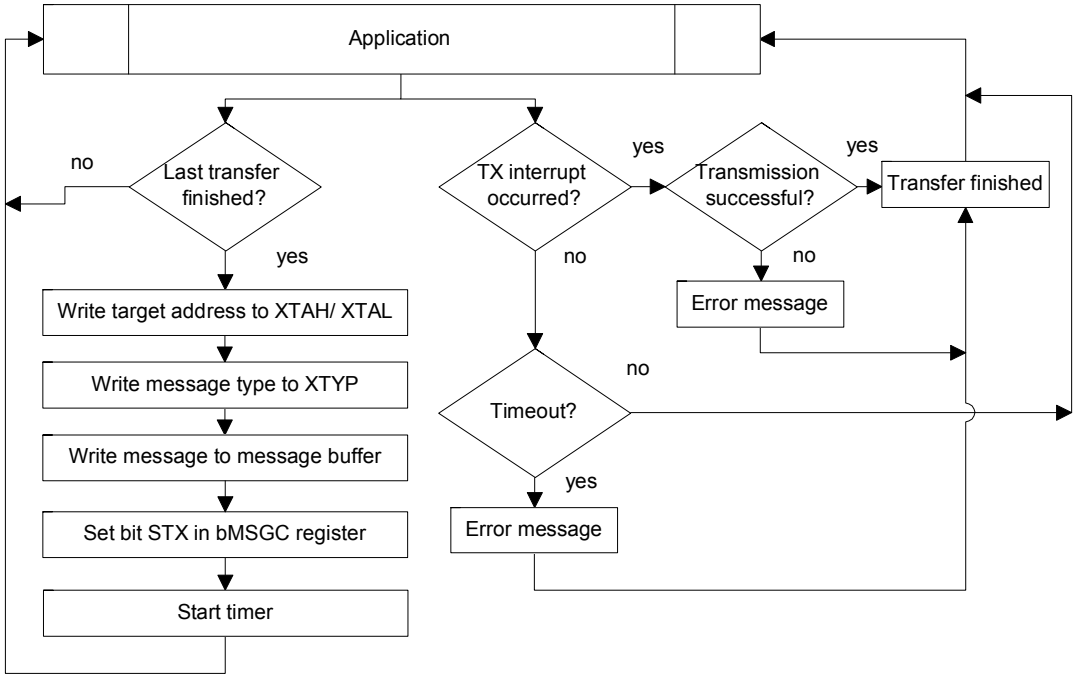


Figure 3-24: Sending a message on the control channel (Low level).

For more detailed information, please refer to [7].

3.3.8.2 Receiving Messages

An incoming message is recognized by checking the “Message Received Bit” in the message status register. This can be done either by polling, or via interrupt.

If an incoming message was recognized, the “Message Receive Section” must be read by the external micro controller. This section contains the received message type, the message itself, and the sender’s address.

After reading the message, the receive buffer must be unlocked with the help of the “Receive Buffer Enable Bit” (RBE) in the message control register. This mechanism avoids the loss of data for incoming messages at a high frequency.

For more detailed information, please refer to [7].

3.3.8.3 Acknowledgement and Data Security

The transmission of control data is secured with the help of a CRC. If CRC check fails on a received message, this is reported to the sending transceiver by an error mechanism. This mechanism repeats the sending of the failed message until either the transmission is successful or the maximum number of retries (which can be set by the application) has been reached.

If the maximum number of retries is reached, this is reported to the application by an error message. Depending on the definition of the external network management, the message may be resent by the application, eventually with a changed maximum number of retries. Re-initialization of the device that does not respond is another possibility.

Another mechanism for securing transmission of control data is a handshake between the external micro controller and the MOST Transceiver on incoming messages. Another message can be received only if an incoming message was read, and the receive buffer of the transceiver is unlocked by the micro controller. If another node tries to send a message to a node with a locked receive buffer, this message is rejected by the receiving transceiver. The same retry mechanism is activated as used on CRC errors. How the external network management reacts to an error of this kind must be defined for this case.

For more detailed information, please refer to [7].

3.3.9 Remote Control

Remote control is a special functionality of the MOST Transceiver. It does not influence the transmit buffer, nor the receive buffer of the remote controlled device, that is, its application stays untouched by this operation. For more detailed information, please refer to [7].

3.3.9.1 Remote Read Message

In order for another node to read the memory area of a MOST Transceiver, a special kind of message must be used. This message is different in terms of the message type written into the XTYP register (0xC1). Here, the value 0x01 (Remote read) must be stored. XMIT control data byte 1 (located at 0xC5) contains the address from which data should be read. Beginning at this address, 8 bytes will always be read. After having sent the message, a transmit interrupt is released if enabled. If the transmission is marked as successful, the result can be read from the XMIT control data bytes 3 through 10 (0xC7 .. 0xCE).

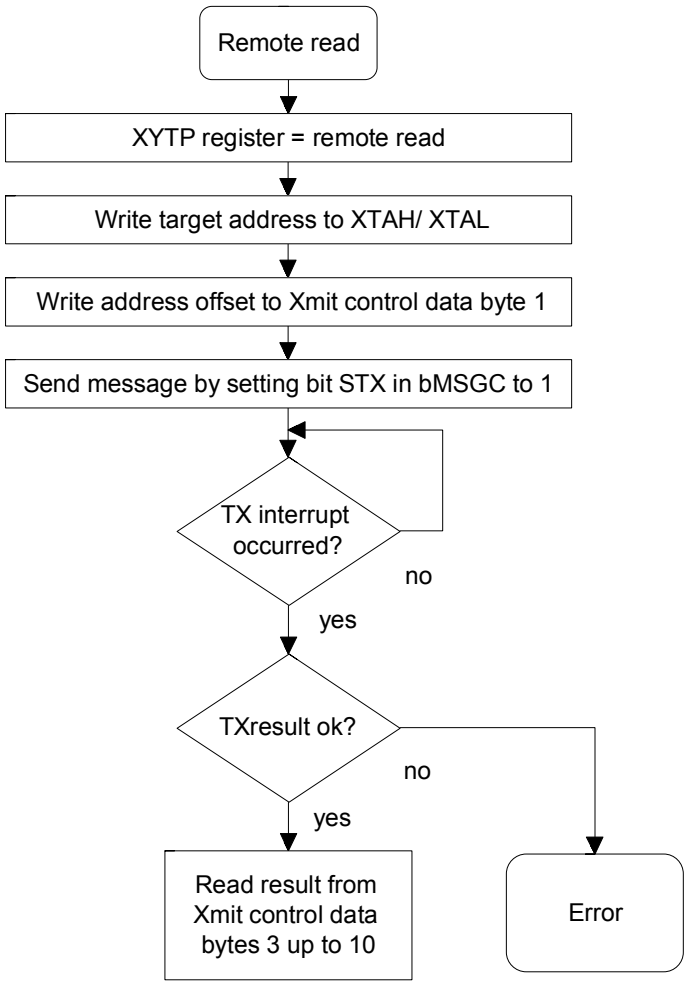


Figure 3-25: Remote read.

3.3.9.2 Remote Write Message

The remote controlled writing to registers of other MOST Transceiver chips is done in principle as it is on the remote read access. Compared with remote read, the telegram is extended by the number of bytes to write. This information is written to register XMIT control data byte 2 (0xC6). Message type (XTYP) Remote Write (0x02) must be used. The data must be written to the XMIT control data bytes 3 through 10 (0xC7 .. 0xCE). Up to eight data bytes can be written per remote write access.

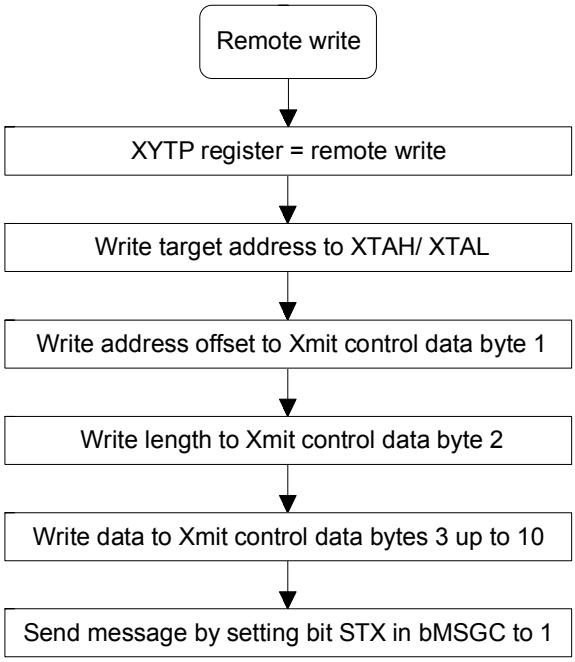


Figure 3-26: Remote write.

3.4 Handling Synchronous Data

3.4.1 MOST NetServices (Application Socket)

The MOST Transceiver already provides convenient mechanisms for administrating synchronous channels. In addition to that, the NetServices have routines that use those mechanisms to provide simple and flexible access to the synchronous resources of the network.

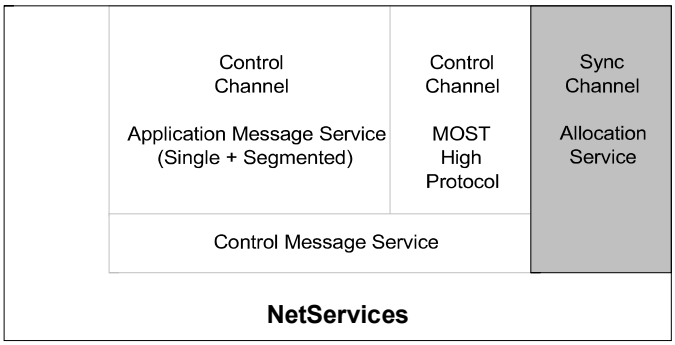


Figure 3-27: NetServices for the synchronous channel

On the network, 60 bytes for synchronous and asynchronous transport are available per frame. A certain number of these channels can be used for synchronous data transfer. Here, several channels can be clustered to a synchronous connection for an application. Every connection that does not use more than 8 channels gets a single handle (Connection Label), which is identical to the lowest number of used channels.

Access to the channels within a device (putting data onto the channels, or getting data from the channels) is done through the source data ports of the MOST Transceiver in several different modes. Connecting the source ports with the channels is controlled via the Routing Engine (RE).

The following routines are implemented in the NetServices:

- **SyncAlloc (Source):**
  Allocating channels for synchronous data transfer and routing the data from the source data port of the MOST Transceiver to these channels via RE.
- **SyncDealloc (Source):**
  Restoring RE, i.e., direct routing of the formerly-occupied channels from input to the output of the MOST Transceiver, and then de-allocation of the channels.
- **SyncOutConnect (Sink):**
  Routing of the channels specified in the function call from the input of the MOST Transceiver to the source ports of a sink (also specified in the function call).
- **SyncOutDisconnect (Sink):**
  Restoring RE in a sink, i.e., removing the connection between the synchronous channels on the network and the source data ports of the MOST Transceiver.

- **SyncFindChannels:**
  Seeking channels belonging to a certain handle.
- **MostSetBoundary (Timing Master):**
  Modifying the boundary between synchronous and asynchronous data transfer.

3.4.1.1 Basic Functions on Application Level

On the application level, different basic functions in sources and sinks are realized, which serve the administration of synchronous connections. They themselves access the routines of the NetServices. The synchronous data transfer does not need to be modeled functionally.

3.4.1.1.1 NetBlock

NetBlock contains the function **SourceHandles**, by which a controller can learn which function block in the Slave holds a connection. Therefore it asks:

```
Controller -> Slave: NetBlock.Pos.SourceHandles.Get (Handle)
```

and gets the function blocks that work with the respective handle in the answer. There can be multiple function blocks in a device:

```
Slave -> Controller: NetBlock.Pos.SourceHandles.Status (Handle, FBlockID.InstID,
                                                        Handle, FBlockID.InstID,
                                                        ...)
```

In case the handle is not used, the following error is reported:

```
Slave -> Controller: NetBlock.Pos.SourceHandles.Error (0x07, 0x01, Handle)
```

If the controller specifies 0xFF as handle, it gets the handles of all connections used in the device, and the IDs of the function blocks using them.

FUNCTIONS				
FktID	**OPType**	**Sender**	**Receiver**	**Explanation**
SourceHandles	Get	Controller	NetBlock	Requesting function blocks working with a certain handle.
	Status	NetBlock	Controller	Answer

Table 3-17: Functions in NetBlock in conjunction with the administration of synchronous resources.

3.4.1.1.2 Function Block

Also in a function block that contains an application, different functions for administering synchronous connections are implemented. By using function SyncDataInfo it can be requested, for how many connections the function block may serve as source (parameter SourceCount, 8 bits), or as sink (parameter SinkCount, 8 bits):

```
Controller -> Slave: FBlockID.InstID.SyncDataInfo.Get

Controller -> Slave: FBlockID.InstID.SyncDataInfo.Status (SourceCount, Sink-
                                                          Count)
```

Functions				
FktID	**OPType**	**Sender**	**Receiver**	**Description**
SyncDataInfo	Get	Controller	Function block	Request of the number of synchronous sources and sinks within a function block.
	Status			Answer

Table 3-18: Common functions in a function block for administering synchronous resources

3.4.1.1.2.1 Synchronous Source

Property **SourceInfo** contains more detailed information about the kind of synchronous source data. On a request with the SourceNr (8 bits starting at 0x01; if a function block has only a single source, it always has 0x01):

```
Controller -> Slave: FBlockID.InstID.SourceInfo.Get (SourceNr)
```

one receives

```
Slave -> Controller: FBlockID.InstID.SourceInfo.Status (SourceNr, DataType,
                                                        [DataDescription])
```

The parameter **DataType** describes the kind of synchronous data stream that is sent by the source. Depending on DataType, a DataDescription may follow. The following synchronous data types are defined at the moment:

Data type 0x00 Audio:

Here, the additional parameters Resolution, AudioChannels, Delay and Handle/Channels are specified.

```
DataDescription = Resolution, AudioChannels, SrcDelay, Channels
```

Parameter **Resolution** (1 Byte) specifies the resolution of audio samples in bytes. Parameter **AudioChannels** (1 Byte) specifies the number of audio channels, e.g., 1 for mono, 2 for stereo etc. Parameter **SrcDelay** (1 Byte) specifies the delay of the synchronous data with respect to the Timing Master. It is equal to the contents of the node delay register (NDR) of the MOST Transceiver. In the last parameter, the single **Channels** (1 byte per channel) are listed. The first channel corresponds to the handle. If the source has not allocated channels at the moment, it returns 0xFF.

MOST Transceiver is able to receive many different audio formats and convert them to its raw data format, or to generate many different audio formats from the transported raw data. For audio transmissions, the following minimum appointments are valid:

- Audio-NF will be transported CD-DA compatible (Compact Disk Digital Audio)
- The sequence of channels is: Front left, front right, rear left, rear right. The most significant byte is transmitted first.

Examples:

16 Bit Stereo: Resolution = 0x02, Channels = 0x02,
Sequence on the bus : MSB left, LSB left, MSB right, LSB right.

24 Bit Stereo: Resolution = 0x03, Channels = 0x02,
Sequence on the bus : MSB left, central byte left, LSB left, MSB right, central byte right, LSB right.

If the property SourceInfo is not implemented by the source, data type audio is assumed by default.

Data type 0x01 CD ROM:

This data type describes CD-ROM raw data before being processed by a CD-ROM decoder. This data might be of type audio, or CD-I, or Video-CD respectively.

```
DataDescription = Blockwidth, Channels
```

For data type CD-ROM, parameter “Blockwidth” is transmitted. It specifies the number of transmitted bytes per MOST frame.

Examples:

Single Speed CD : Blockwidth = 0x04
Double Speed CD : Blockwidth = 0x08

Per Default, Blockwidth = 0x04 will be assumed.

In property **SourceName**, a name for the synchronous source data can be requested:

```
Controller -> Slave: FBlockID.InstID.SourceName.Get (SourceNr)
```

As an answer, a string is returned that contains the required name:

```
Slave -> Controller: FBlockID.InstID.SourceName.Status (SourceNr, SourceName),
```

The source is induced to allocate synchronous channels by function **Allocate** :

```
Controller -> Slave: FBlockID.InstID.Allocate.StartResult (SourceNr)
```

The function block accesses the respective routines of the NetServices (SyncAlloc as instruction, and SyncAllocComplete as answer) and reports the following result on success:

```
Slave -> Controller: FBlockID.InstID.Allocate.Result (SourceNr,
                                                      SrcDelay, Channels),
```

Error handling:

If the allocation was not successful due to a lack of enough free channels, an error code is generated with error code “Function specific” and as Error Info the SourceNr and the required channels containing 0xFF (not allocated channels):

```
Slave -> Controller: FBlockID.InstID.Allocate.Error (SourceNr, Channels),
```

DeAllocate induces the source to de-allocate allocated synchronous channels:

```
Controller -> Slave: FBlockID.InstID.DeAllocate.StartResult (SourceNr)
```

The function block accesses the respective routines of the NetServices (SyncDealloc as instruction and SyncDeallocComplete as answer) and sends the result:

```
Slave -> Controller: FBlockID.InstID.DeAllocate.Result (SourceNr)
```

Some synchronous source applications require to either start or stop transmission of stream data controlled by superior layers. In general, there are specific functions that need to be called for performing the starting or stopping. It is an advantage, if the respective controller does not need to know about any of those specific functions. Therefore, for every synchronous source data stream, the abstract method SourceActivity is defined:

```
Controller -> Slave: FBlockID.InstID.SourceActivity.StartResult (SourceNr,
                                                                 Activity)
```

Through parameter Activity = [On/Off/Pause], synchronous data transfer can either be started, stopped, or paused. After completion of the respective action, the following reply will be generated:

```
Slave -> Controller: FBlockID.InstID.SourceActivity.Result (SourceNr, Activity)
```

As an example, a CD player could be set to mode Play by SourceActivity.Start(On). Please note, that the allocation of channels requires the calling of function Allocate before. Allocation is not handled by SourceActivity. Calling SourceActivity.Start(Pause) sets the player to Pause mode.

FUNCTIONS				
FktID	**OPType**	**Sender**	**Receiver**	**Explanation**
SourceInfo	Get	Controller	Application of the source	Request of information about the kind of synchronous source data
	Status	Application of the source	Controller	Answer with parameters depending on type
SourceName	Get	Controller	Application of the source	Requesting the name of the synchronous source data
	Status	Application of the source	Controller	Answer as string
Allocate	StartResult	Controller	Application of the source	Demand for allocation
	Result	Application of the source	Controller	Reply with result of allocation
DeAllocate	StartResult	Controller	Application of the source	Demand for de-allocation
	Result	Application of the source	Controller	Reply with result of de-allocation
SourceActivity	Start	Controller	Application of the source	Starting, stopping, or temporary stopping of a synchronous source

Table 3-19: Functions in a function block with a synchronous source, in conjunction with administering synchronous resources.

3.4.1.1.2.2 Synchronous Sink

In a function block that is used as a sink for synchronous data, analog functions like those for a source are implemented. Error handling is also done in an analogous way. Via function **SinkInfo**, information about which kind of data the sink can handle is stored. Here the SinkNr specifies the desired sink in the function block (8 bits starting at 0x01; if a function block has only a single sink, it always has SinkNr 0x01):

```
Controller -> Slave: FBlockID.InstID.SinkInfo.Get (SinkNr)

Slave -> Controller: FBlockID.InstID.SinkInfo.Status (SinkNr, DataType,
                                                      [DataDescription])
```

The parameters are identical to those described above, except that SrcDelay must be replaced by SinkDelay.

SinkName is available to request a name for the synchronous data:

```
Controller -> Slave: FBlockID.InstID.SinkName.Get (SinkNr)

Slave -> Controller: FBlockID.InstID.SinkName.Status (SinkNr, SinkName),
```

Function **Connect** induces the sink to connect to certain channels:

```
Controller -> Slave: FBlockID.InstID.Connect.StartResult (SinkNr, SrcDelay,
 Channels)
```

It accesses the respective routine of the NetServices (SyncOutConnect). SrcDelay is the delay that is passed to the sink to provide the possibility of delay compensation. The sink returns as result:

```
Slave -> Controller: FBlockID.InstID.Connect.Result (SinkNr),
```

Function **DisConnect** induces the sink to remove a certain connection:

```
Controller -> Slave: FBlockID.InstID.DisConnect.StartResult (SinkNr)
```

It accesses the respective routine of the NetServices (SyncOutDisconnect) and returns as result:

```
Slave -> Controller: FBlockID.InstID.DisConnect.Result (SinkNr),
```

In many cases, the output of synchronous data on a sink must be stopped. For this, the function **Mute** is used:

```
Controller -> Slave: FBlockID.InstID.Mute.Set (SinkNr, Status)
```

with Status = [On/Off].

In smaller systems it might be useful to use a generic peer-to-peer approach for establishing connections. Therefore the sink is provided with function **ConnectTo**:

```
Controller -> Slave: FBlockID.InstID.ConnectTo.StartResult (FBlockID.InstID.
                                                            SourceNr)
```

With this method, the sink is induced to connect to a certain source. The sink uses SourceInfo to check whether the source sends a signal which it can handle. Then source and sink connect without external influence. On success the sink reports:

```
Controller -> Slave: FBlockID.InstID.ConnectTo.Result (FBlockID.InstID.
                                                       SourceNr)
```

FUNCTIONS				
FktID	**OPType**	**Sender**	**Receiver**	**Explanation**
SinkInfo	Get	Controller	Application of the sink	Request of information with respect to the kind of the synchronous data the sink can handle
	Status	Application of the sink	Controller	Answer with parameters depending on type
SinkName	Get	Controller	Application of the sink	Requesting the name of the synchronous data the sink can handle
	Status	Application of the sink	Controller	Answer as string
Connect	StartResult	Controller	Application of the sink	Demand for connecting to certain channels
	Result	Application of the sink	Controller	Reply with result
DisConnect	StartResult	Controller	Application of the sink	Demand for removing a connection
	Result	Application of the sink	Controller	Reply with result
Mute	Set	Controller	Application of the sink	Starting/ stopping the output of the synchronous data
ConnectTo	StartResult	Controller	Sink	Demand for building a connection to a certain source
	Result	Sink	Controller	Reply with result

Table 3-20: Functions in a function block with a synchronous sink in conjunction with the administration of synchronous resources.

The basic functions described to this point provide the detailed requesting of sources and sinks for information about source data that can be handled. This is the basis for a flexible, self-configuring system. It is possible to implement a peer-to-peer approach, where sink and source act more independently from controlling devices:

> Only source and sink must have information about the handling of their own data types. The HMI or a higher controller does not need to handle some types of information, e.g., SourceInfo. A data sink checks whether the data type of the source matches its own data type before building the connection. This means that data sinks need to know only the parameters of their own data types, which allows the data type definition to be extended easily if required.

3.4.2 MOST NetServices (Basic Layer)

When using the basic layer of the MOST NetServices, the allocation and de-allocation of channels for transporting synchronous source data is possible, but extra functions are required to inform the NetBlock of allocation results.

3.4.3 Direct Access to OS8104

3.4.3.1 Serial Interface

Up to 4 source data ports are available for input and output of synchronous source data in the MOST Transceiver. Each source data port can handle up to 64 bits per frame. To achieve higher data rates, there is the possibility of cascading several source data ports. Nevertheless, the number of source data ports is decreased by this.

Every port has the ability to handle different formats, which can be specified by control registers. For more detailed information please refer to [7].

3.4.3.2 Parallel Interface

In addition to the ability to input or output source data in serial mode, the MOST Transceiver also has a parallel interface. Asynchronous or control data can also be input or output via this parallel interface.

A FIFO method provides buffering for the purpose of external synchronization. Data flow is handled via the respective control pins of the chip.

3.4.3.3 Compensating Network Delay

Every active node in the ring (source data bypass inactive), generates 2 samples delay for the source data (caused by internal processing). Especially in top HIFI applications, this is an unpleasant effect. The MOST system therefore provides mechanisms that allow compensation for this delay. Every chip is provided with the information about the general delay of the entire system $\Delta T_{Network}$, and the delay up

to its own node ΔT_{Node} with respect to the Timing Master. In addition to that, the delay of the active source device ΔT_{Source} must be made available by control messages.

Based on that information, the delay ΔT_{comp}, which must be compensated for, can be calculated with the help of the formula below:

$\Delta T_{comp} = \Delta T_{Node} - \Delta T_{Source} - 2\,[Samples]$ for $\Delta T_{Source} < \Delta T_{Node}$

$\Delta T_{comp} = \Delta T_{Network} - \Delta T_{Source} + \Delta T_{Node} - 2\,[Samples]$ for $\Delta T_{Source} > \Delta T_{Node}$

(ΔT_{xxx} ≡ Contents of the respective register in the chip * 2 Samples Delay)

3.5 Handling Asynchronous (Packet) Data

3.5.1 Direct Access to OS8104

A data packet that can be sent in the asynchronous area consists of 48 bytes of data and a 16 bit receiver address (logical node address only):

The data field is significantly longer than that of the control channel. Unlike the control channel, no ACK/NAK mechanism or low level retries are implemented, since they are not necessary for most of the applications. Nevertheless, the telegram is checked. A transport protection must be implemented on a higher level.

3.5.1.1 Priorities

In every node (MOST Transceiver) the priority for packet data transfer can be specified. It is based on the control of access authorization of a node on free frames in the data stream on the bus:

Range:	0x1..0x7
Default:	0x1 (Highest Priority)
Register:	bPPI

At first, only priority 0x1 is used.

3.5.2 MOST NetServices

The NetServices have a mechanism, called the Asynchronous Data Transmission Service, by which the data packets described above can be sent and received. It handles the respective registers in the MOST Transceiver.

3.5.2.1 Securing data

For stream data of high bandwidth, e.g., graphical applications, it is not useful to implement mechanisms for secure data transmission. On one hand the data security (bit error rate) of MOST is approximately 10^{-12}, on the other hand every securing mechanism must be checked by a μController, which becomes more and more difficult, as the bandwidth is increased. In case of errors, transmissions would be repeated, which would cause delays that might be unwanted. It must be decided for each application, whether mechanisms for secure data transmission would be useful, and if so, which implementation to use.

For certain applications which transmit in the asynchronous area at a lower bandwidth, it might be useful to implement securing mechanisms. The telegram structure quite alike that of the control channel could be used by setting TelID to 4 bits and TelLen to 12 bits.

Data area MOST Transceiver = 48 Byte (Data Link Layer 48 Bytes Mode)

16 Bit	8 Bit	8 Bit	12 Bit	4 Bit	4 Bit	12 Bit	8 Bit	8 Bit		8 Bit
DeviceID	FBlock ID	Inst. ID	Fkt ID	OP Type	Tel ID	Tel Len	Data 0	Data 1	...	Data 41

Data area MOST Transceiver = 1014 Byte (Alternative Data Link Layer)

16 Bit	8 Bit	8 Bit	12 Bit	4 Bit	4 Bit	12 Bit	8 Bit	8 Bit		8 Bit
DeviceID	FBlock ID	Inst. ID	Fkt ID	OP Type	Tel ID	Tel Len	Data 0	Data 1	...	Data 1007

TelID: Identification of kind of telegram

Meaning	TelID
MOST High Protocol User data	8
MOST High Protocol Control data	9
Reserved for Ethernet frames	A

TelLen: = up to 1008 (42)
Specifies the length of the data field, i.e. ,the number of bytes after TelLen

Data X: Data bytes

For securing data, MOST High Protocol is used here. For more detailed information, please refer to [3].

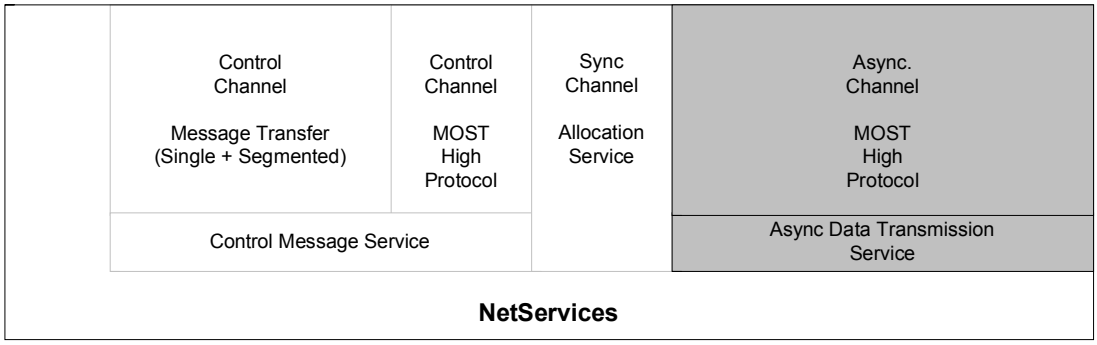


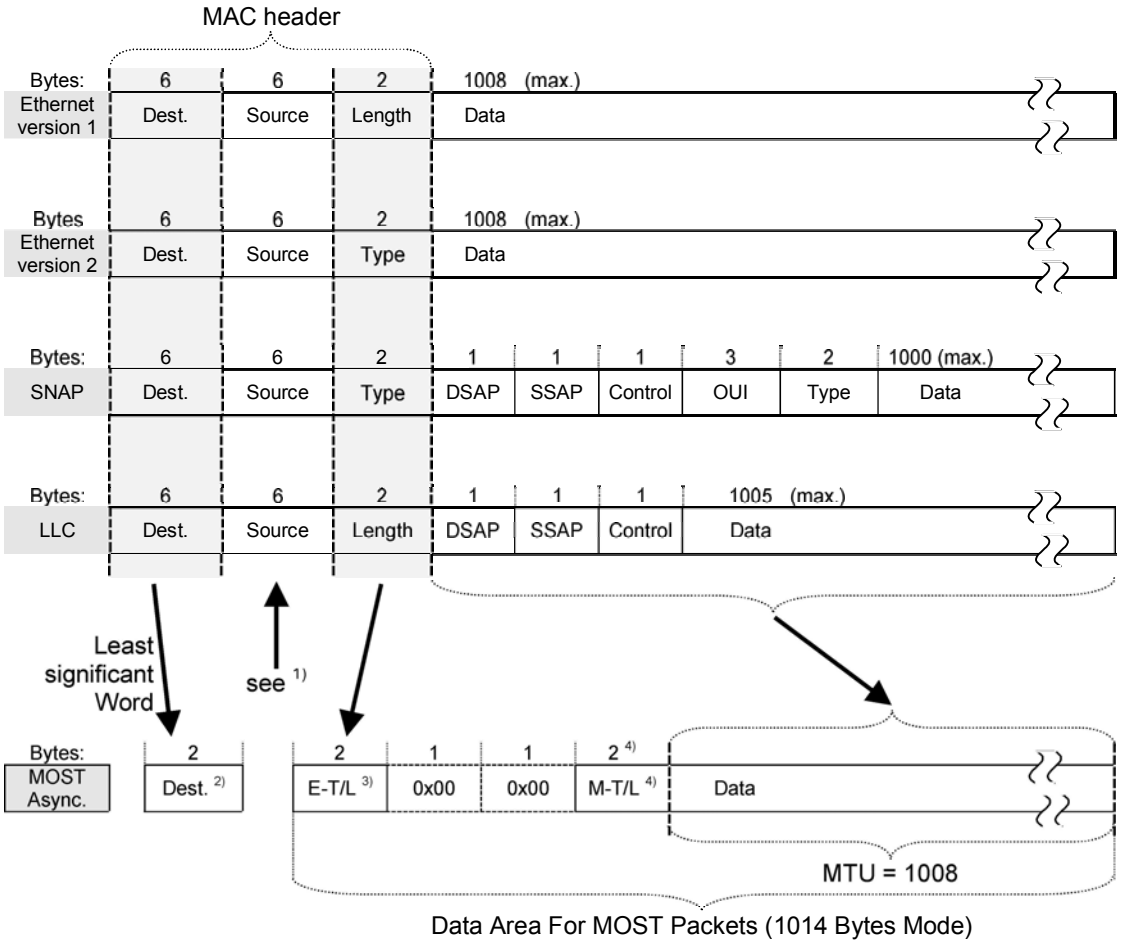
Figure 3-28: NetServices: Services for the asynchronous channel.

3.5.3 MOST Asynchronous Medium Access Control (MAMAC)

For being able to run commonly used network protocols like TCP/IP (including IPX, NetBEUI, ARP) through the Asynchronous (Packet Data) channel of MOST, MOST Asynchronous Medium Access Control (MAMAC) was defined. The following sections show, how the structures of Ethernet are transported through MOST, and how addressing is done.

3.5.3.1 Packaging Frames

Due to the structures shown below, it is possible to transport Ethernet version 1, version 2, SNAP, and LLC type frames through MOST. The MTU value for MOST is 1008 bytes per frame.



Data Area For MOST Packets (1014 Bytes Mode)

Notes:
1. The Source Specifier is handled by the MOST Network automatically. No special treatment required.
2. The least two bytes of the Ethernet Destination address are used for addressing on MOST network.
3. Ethernet Length or Type specifier
4. MOST Telegram ID and Length specifier. Telegram ID requires 4Bits (value is 0xA), while the Length specifier uses 12 Bits.

Figure 3-29: Ethernet frames versus MOST Packets

In Figure 3-29, Preamble, Start-of-frame and Checksum fields have been removed from the original Ethernet frame. This is appropriate, since they do not need to be transported through MOST.

3.5.3.2 Addressing Scheme

3.5.3.2.1 Address Generation

There are several possible ways to stipulate unique addresses for the MOST asynchronous packet transfer. Some mechanisms are described in this MOST specification. The mechanisms described below, are examples only. Every address configuration is possible, as long as the addresses are unique within the MOST network. Available approaches are :

- Using the Node position of the MOST chips for address creation (a common approach, as described in section 3.3.1 on page 144).
- Using factory-set addresses.
- Manual or automatic address assignment after the system has been assembled

The MOST Transceiver OS8104 provides two address registers for packet data transfer:

- The logical standard address which is identical with the address used for the control channel (Logical Node Address; *Rx/TxLog*).
- The alternative packet address which can be used for the broadcast function.

The address used for Ethernet communication must be identical with the address used on Control Channel and for the MOST High protocol. This has no side effects on the NetServices or on MOST High Protocol, since the functional addressing mechanism of NetServices will work with any kind of address as long as they are unique.

At the moment, two address registers are available for Packet Data Transfer. This is based on a property of OS8104. Currently, there no restrictions are planned with respect to the number of address registers. Even today, it is possible to increase the number of usable addresses by using Physical (Parallel Combined) mode of OS8104.

Please note:
Since the address ranges 0x0300 .. 0x03FF and 0x0400…0x043F are used for Group-, Broadcast- and Node-position-Addressing in MOST networks, they should be omitted.

3.5.3.2.2 MAC Address Generation

Some Network stacks expect a 48bit Network Address (MAC Address) for Ethernet networking. In a MOST environment, the MAC addresses are initialized to 00:00:00 first. After reception of Configuration.Status (OK) (network is in a stable state), the MAC address will be calculated. Therefore, the logical device address is copied to the least 16 bits of the MAC address:

```
00:00:00 -> 00:01:23 (If the logical address of the device is 0x0123)
```

All unused digits shall stay at 0x0. As an alternative approach, they could contain the Organizationally Unique Identifier or 'company_id' of the MOST system integrator, which is assigned by IEEE.

Please note:
MAC address is valid only on occurrence of Configuration.Status (OK).

3.5.3.2.3 Handling Broadcast

The Ethernet broadcast address is 0xFFFFFFFFFFFF. This address can be mapped to 0xFFFF in MOST, to allow broadcast of packets over the MOST packet channel. On the receiving side, 0xFFFF is finally extended to Ethernet broadcast 0xFFFFFFFFFFFF again.

3.6 Controlling Synchronous/Asynchronous Bandwidth

When administrating the boundary between synchronous and asynchronous data area, two contrary requirements must be taken into consideration. On one hand, there should be as much bandwidth as possible for asynchronous data transfer, so it is not reserved for unused synchronous channels. On the other hand, the boundary should be changed only in rare cases, since all synchronous connections must be re-allocated after the boundary was changed.

Even with the most adverse usage of a fully equipped vehicle, the entire available bandwidth for synchronous transfer is seldom used. Generally, less bandwidth is required for synchronous transfer.

System initialization adjusts the boundary to the center of the bandwidth. During runtime it is shifted only “to the right”, that is, in the direction of an extension of the synchronous area, up to a limit, which will reserve, e.g., one quadlet for asynchronous data transfer. There is no shifting to the left. The boundary is returned to its default value only after a re-initialization of the system.

The changing of the boundary between synchronous and asynchronous area is done physically by the Timing Master, located in the system Master device. Timing Master functionality is provided by a MOST Transceiver, so the NetworkMaster therefore provides function **Boundary**. It is controlled by the ConnectionMaster:

```
??? -> SystemmasterDevice: NetworkMaster.0.Boundary.Set (BoundaryDescriptor)
```

The parameter BoundaryDescriptor corresponds to the one used in the MOST Transceiver, and can be written directly into that register by the NetworkMaster.

The current status of the boundary can be requested by:

```
??? -> SystemmasterDevice: NetworkMaster.0.Boundary.Get
```

The answer is:

```
SystemmasterDevice -> ???: NetworkMaster.0.Boundary.Status
(BoundaryDescriptor)
```

FUNCTIONS				
FktID	**OPType**	**Sender**	**Receiver**	**Explanation**
Boundary	Set	Connection Master	NetworkMaster	Setting of boundary between synchronous and asynchronous area
	Get	Controller	NetworkMaster	Request for status of boundary between synchronous and asynchronous area
	Status	Network Master	Controller	Answer on request

Table 3-21: Functions for administration of boundary between synchronous and asynchronous area in NetworkMaster.

3.7 Connections

3.7.1 Synchronous Connections

3.7.1.1 Administering (ConnectionMaster)

For managing synchronous connections in complex networks it might be useful to introduce a higher control instance on the application level. It will be implemented in function block ConnectionMaster. Since the ConnectionMaster must verify that a requested connection does not already exist, all requests for establishing connections must be directed to the ConnectionMaster. For building a point-to-point connection, it provides the method **BuildSyncConnection**:

```
Controller -> ???: ConnectionMaster.0.BuildSyncConnection.StartResultAck
                                        (SenderHandle, Source, Sink)
```

The question marks are used in this example to indicate any device, since it is not relevant which device the ConnectionMaster is implemented in. Source stands for the source of a connection with:
Source = FBlockID.InstID.SourceNr.

Sink is the sink of the connection with:
Sink = FBlockID.InstID.SinkNr.

After a successful connection, the ConnectionMaster returns:

```
??? -> Controller: ConnectionMaster.0.BuildSyncConnection.ResultAck
                                         (SenderHandle, Source, Sink)
```

Error handling:

If the connection fails, the ConnectionMaster answers with OPType “ErrorAck” (0x9) and the ErrorCode “ProcessingError” (0x42), and returns the parameters Source and Sink:

```
??? -> Controller: ConnectionMaster.0.BuildSyncConnection.Error
                                                    (SenderHandle)
```

Removing a connection is done in an analogous way, with the help of the **RemoveSyncConnection** method.

The ConnectionMaster generates an array of all existing connections including sources and sinks, where it adds more information. This array is accessible in function **SyncConnectionTable** :

```
??? -> Controller: ConnectionMaster.0.SyncConnectionTable.Get

Controller -> ???: ConnectionMaster.0.SyncConnectionTable.Status
                   (Source, Sink, SrcDelay, Channels,
                    Source, Sink, SrcDelay, Channels,
                    Source, Sink, SrcDelay, Channels,...)
```

The parameters are the same as those described above. SyncConnectionTable can not be set directly. Building and removing connections is done only with BuildSyncConnection and RemoveSyncConnection.

After switching off the network (light off), the contents of SyncConnectionTable are deleted, leaving no synchronous connections in the system. They must be rebuilt by new requests of the initiator(s).

FUNCTIONS				
FktID	**OPType**	**Sender**	**Receiver**	**Explanation**
BuildSyncConnection	StartResultAck	Controller	Connection Master	Request for building connection
	ResultAck	Connection Master	Controller	Answer with result
SyncConnectionTable	Get	Controller	Connection Master	Request of that property, where the ConnectionMaster stores all active point-to-point connections
	Status	Connection Master	Controller	Answer
RemoveSyncConnection	StartResultAck	Controller	Connection Master	Request for removing connection
	ResultAck	Connection Master	Controller	Answer with result

Table 3-22: Functions in ConnectionMaster in conjunction with the administration of synchronous connections.

3.7.1.2 Establishing Synchronous Connections

In building a synchronous connection, the ConnectionMaster induces the source to allocate the required channels. After a positive answer from the source about the allocation of channels, the ConnectionMaster can pass the data for building the connection to the sink. This mechanism is explained in the following figure, and the text below.

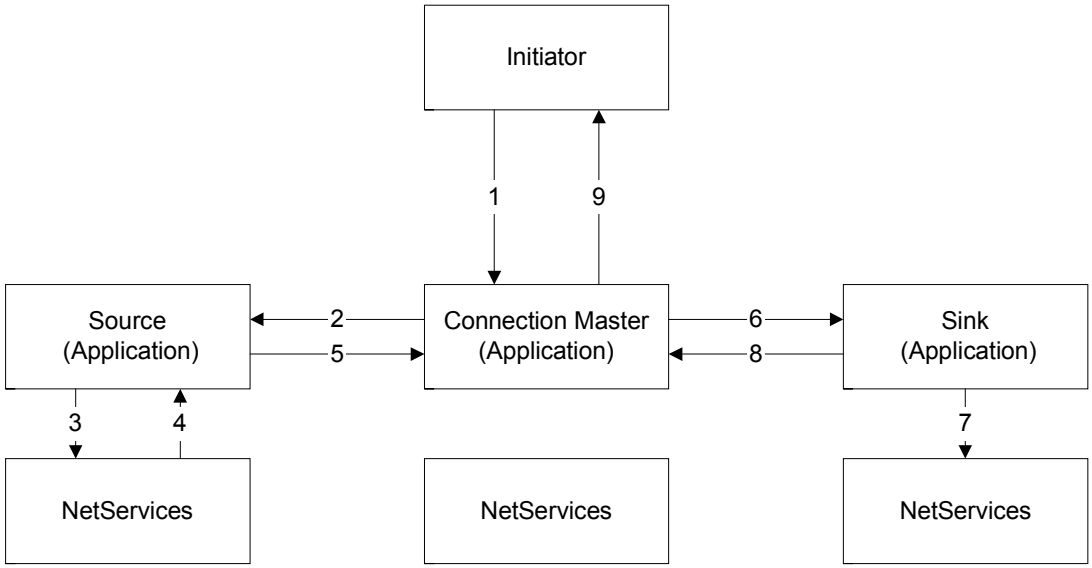


Figure 3-30: Flow of building a connection in synchronous area.

Step by step description:

1. Method *BuildSyncConnection* is started by an initiator (e.g., a HMI), for building a connection between a source and a sink.
2. If this source has not (yet) allocated a transfer channel, the ConnectionMaster starts method *Allocate* for allocating a transfer channel, otherwise continue at step 6.
3. The application of the source calls routine *SyncAlloc* of the NetServices.
4. The NetServices returns the result to the application of the source by Callback Function *SyncAllocComplete*.

5. The application of the source processes the result:
   - ALLOC_GRANT: Enough free channels.

     Reply to ConnectionMaster as *Allocate.Result* with parameters SourceNr, SrcDelay and Channels.
   - ALLOC_BUSY: Timing Master is busy on processing other allocation requests.

     The source retries allocation (maximum 5 retries with 10ms delay).
   - ALLOC_DENY: Not enough channels.

     Reply to ConnectionMaster as *Allocate.Error* with parameters SourceNr and Channels, where Channels refers to the needed channels, the values however are set to 0xFF (e.g., 0x FF FF FF FF for 4 channels); continue at step 6.
6. If the result is *Allocate.Result*, the ConnectionMaster starts method *Connect* of the sink, for routing data to the respective channel. For simplification, it passes the parameters Channels and SrcDelay of the source.
7. If the ConnectionMaster receives the message Allocate.Error from the source, it can initiate moving the boundary between the synchronous and the asynchronous area. It writes the new value of the boundary descriptor to function Boundary of the function block NetworkMaster. Please note that all synchronous connections must be re-built after boundary was changed.
8. The application of the sink starts routine *SyncOutConnect* of the NetServices.
9. The sink reports the routing to the ConnectionMaster as *Connect.Result*.
10. The ConnectionMaster reports the result of establishing the connection to the initiator by using *BuildSyncConnection.ResultAck*. If the building of the connection was successful, the ConnectionMaster internally stores the connection data. This way, if another sink is connected to this source, only the allocation data needs to be sent to the new sink. On failure, an error handling must be performed (e.g., displaying a message).

Some important connections are established once during system initialization, if the devices are available. The connecting is initiated by the ConnectionMaster and the connections stay established constantly during operation.

Administration of connections is done only by the ConnectionMaster. Other devices cannot detect whether connections are pre-installed or not. They behave as if there were no fixed connections, i.e. they apply to the ConnectionMaster for establishment or termination of connections.

3.7.1.3 Removing Synchronous Connections

The initiator terminates a connection at the ConnectionMaster. The ConnectionMaster induces the sink to disconnect its routing connection to the transfer channel. After a positive answer, and if the channel is not in use by another sink, the ConnectionMaster induces the source to de-allocate its channels. After that, the ConnectionMaster reports the result of the termination to the initiator.

Step by step description:

1. An initiator starts method *RemoveSyncConnection* in the ConnectionMaster, for terminating a connection between a sink and a source.
2. The ConnectionMaster starts method *DisConnect* in the application of the sink, for disconnecting its routing connection to the transfer channels.
3. The application of the sink starts *SyncOutDisconnect* in its NetServices.
4. The application of the sink reports the result to the ConnectionMaster by *DisConnect.Result.*
5. If no other sink occupies the transfer channel, the ConnectionMaster starts method *DeAllocate* in the application of the source, for releasing the respective channels. Otherwise continue at step 10.
6. The application of the source starts routine *SyncDealloc* in its NetServices.
7. The source evaluates *SyncDeallocComplete* :
   - DEALLOC_GRANT: Successful de-allocation

     Positive answer by *DeAllocate.Result*
   - DEALLOC_BUSY:

     The Timing Master is busy handling other allocation/ de-allocation requests. The source tries another De-Allocation, up to a maximum of 5 retries (after 10 ms each).
8. The source's application answers the ConnectionMaster with *DeAllocate.Result*
9. If the ConnectionMaster has received a positive answer from the source, the respective connection is removed from its internal connection table.
10. By *RemoveSyncConnection.ResultAck*, the ConnectionMaster sends an answer to the initiator, that contains the status of the requested termination of the connection.

3.7.1.4 Supervising Synchronous Connections

If a source that occupies synchronous channels has a defect, it closes its all bypass. By doing that, the source receives incorrect data. In this case, which can be recognized by a short unlock, and/or a NetworkChangeEvent, the reaction will be:

- Every Slave in the network secures its output signals (analog or digital). A loudspeaker mutes, a display shows a default color, etc.

3.8 Timeouts

Name	Affects	Value	Meaning
Initialization			
t_{Config}	Each Device	100 ms	Maximum time between “light on“ at the input of the FOT and the NetOn event.
t_{WakeUp}	Each Device	6 ms	Maximum time between “light on“ at the input of the FOT and “light on“ at the output of the FOT.
$t_{PowerOn}$	Each Device	10 ms	Time that might pass between the switching off of the reset at the MOST Transceiver and the moment when the transceiver generates a Power-On-Interrupt.
t_{Slave}	Timing Slave	2000 ms	Time that might pass (in a Slave device) after initialization of the transceiver, until no more lock errors occur, and the bSBC register contains a value >5 (Ring closed).
$t_{WaitNodes}$	Each active Device	100 ms	Time that might pass, after “Light On“ at the input of the FOT, until an active node has deactivated its bypass.
t_{Lock}	Each Device	100 ms	Time during which no Lock Errors must occur, for being able to declare the Lock as “stable“.
t_{Master}	Timing Master	2000 ms	Time that might pass (in a Master device) after initialization of the transceiver, until no more lock errors occur.
$t_{CfgStatus}$	Network Slave	15000 ms	Time a network Slave will wait after NetOn, for receiving Configuration.Status from NetworkMaster.
t_{Answer}	Network Master	50 ms	Time the NetworkMaster waits for an answer on the request for FBlockIDs.
$t_{WaitAfterNCE}$	Network Master	100 ms	Time the NetworkMaster waits after a NetworkChangeEvent, until it retries to check system configuration.
Shut Down			
$t_{ShutDown}$	Each Device	15 ms	Time after which a device must have switched off its light at the output after a Shut Down event (Light off at the input of the FOT until light off at the output of the FOT).
$t_{Suspend}$	PowerMaster	2000 ms	Time the PowerMaster will wait for a ShutDown.Result (Suspended) after broadcast of ShutDown.Start, until it switches off light.
$t_{ShutDownWait}$	PowerMaster	1000 ms	Time the PowerMaster will wait after a ShutDown.Start (Execute), until it switches off light.
$t_{RetryShutDown}$	PowerMaster	10000 ms	Time the PowerMaster waits after ShutDown.Result (Suspend), until it retries to shut down the network by a new broadcast ShutDown.Start.
Shut Down (To be continued)			

Name	Affects	Value	Meaning
Shut Down (Continued)			
$t_{Runtime}$	PowerMaster	20000 ms	Time, starting at NetOn, the system must have been running before: • The Central Registry can be saved. • Differences to the Registry of the last system run can be saved as error (Error_Registry_New)
Error Management			
$t_{Restart}$	Each Device	300 ms minimum	Time that must been waited for between switching off and switching on again of the network (Light off until Light on at the output of the FOT).
t_{Unlock}	Each Device	70ms	Unlock events with a length greater than t_{Unlock} are handled as errors in the network, otherwise unlocks are reported to the application for getting an eventual local reaction (e.g., Mute). The criterion is not only length, but also the frequency.
$t_{UnlockRecovery}$	Each Device	100ms	Time during which – after lock, no unlocks must occur. If there are no unlocks during that time, the lock can be regarded as stable. This is then reported to the application by the NetServices. The application then can restore the synchronous signals
Ring break diagnosis			
t_{Diag_Master}	Timing Master	28000 ms	Time a Timing Master waits for light at FOT input, during ring break diagnosis.
t_{Diag_Slave}	Slave Device	30000 ms	Time a Slave device waits for light at FOT input, during ring break diagnosis.
t_{Diag_Lock}	Each Device	250 ms	On ring break diagnosis. Time during which there must not be any lock errors, for being able to declare the lock as “stable“. Preliminary value.
Misc.			
$t_{OptPwrLow}$	Each Device	1000 ms	Time during which a device sends at reduced power, after a KWP2000 command. After expiration of this time, the device has to send at maximum optical power again.
$t_{PwrSwitchOffDelay}$	Each Device	5.000 ms	Time after “light off“ at the input, during which an application must stay active, since the light might be switched on again (If the switching off was caused by an error handling by NetworkMaster).
$t_{MsgResponse}$	Each Device	15 ms	Maximum time after which a device must have read a control message from the RX buffer of the MOST Transceiver. This determines the frequency by which the NetServices must be called.

Table 3-23: Timeouts

3.9 Secondary Node

For some applications it can be useful to integrate two MOST Transceiver chips into one device. This section describes, which scenarios are available, and how the tasks are divided up between the two OS8104 chips.

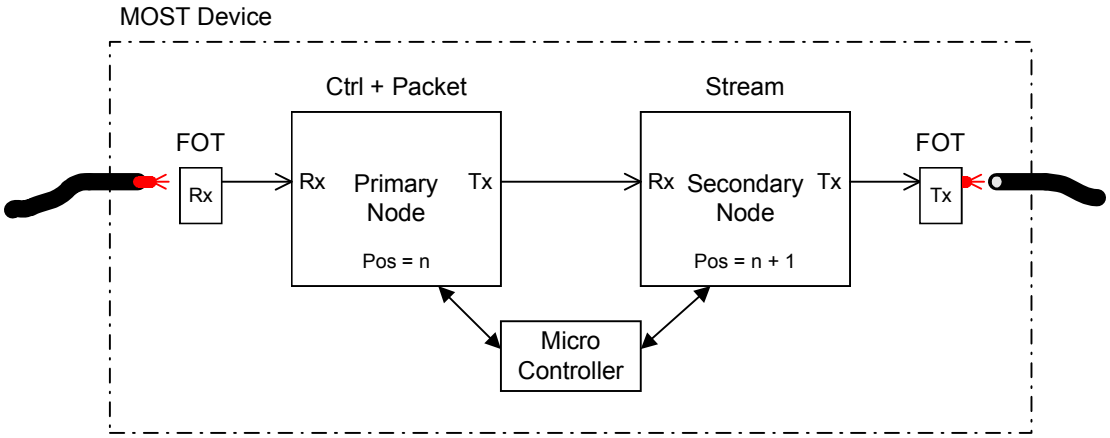
Please note:
The Secondary Node approach applies for Timing Slave nodes only.

When using secondary nodes, both OS8104 are controlled by the same Micro Controller. One of the nodes handles Stream data (Stream), while the other node handles Control Messaging and Packet Data handling (Ctrl + Packet). That node, where Control Messaging and Packet Data transfer are handled (and where the NetServices are running on mainly) is called “Primary Node”. There are two scenarios:

- Scenario 1:
  Primary node : Ctrl + Packet
  Secondary node : Stream
- Scenario 2:
  Secondary node : Stream
  Primary node : Ctrl + Packet

All timing constraints which apply for “normal” MOST nodes (e.g. those with NetServices), must be fulfilled by secondary nodes too. The address of the secondary node should be determined and initialized too.

3.9.1 Scenario 1



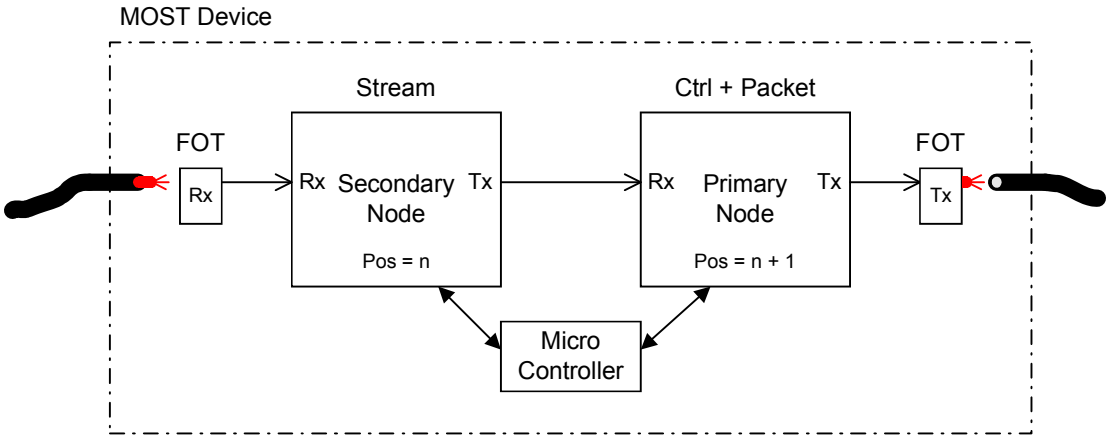
Primary Node
- SP Parallel Async., CP Serial
- SP Parallel Async., CP Parallel

Secondary Node
- SP Serial, CP Serial
- SP Parallel Sync., CP Serial
- SP Parallel Sync., CP Parallel

Figure 3-31: Secondary Node, scenario 1

In Scenario 1, the node position (Pos) of the primary node is always less than the node position of the secondary node. Figure 3-31 shows, which configurations for Source Data Port (SP) and Control Port (CP) are available.

3.9.2 Scenario 2



Secondary Node
- SP Serial, CP Serial
- SP Parallel Sync., CP Serial
- SP Parallel Sync., CP Parallel

Primary Node
- SP Parallel Async., CP Serial
- SP Parallel Async., CP Parallel

Figure 3-32: Secondary Node, scenario 2

In Scenario 2, the node position (Pos) of the secondary node is always less than the node position of the primary node. Figure 3-32 shows, which configurations for Source Data Port (SP) and Control Port (CP) are available.

4 Hardware Section

4.1 Basic HW Concept

The fundamental hardware structure of a MOST Device is displayed in the block diagram below. There are blocks that are not mandatory, since, e.g., a simple MOST Device does not always need a micro controller (active speaker). Areas that are not mandatory are displayed in gray. A MOST Device consists of:

- Optical interface area
- MOST function area
- µC area
- Application area
- Power supply area

A MOST network is awakened optically. All MOST Devices are connected to a continuous power supply. They activate a sleep mode if required. If a device is in sleep mode, the power consumption should be reduced as far as possible (current < 100µA). For this reason, unused areas must be separated from the power supply. Only the sections that are absolutely necessary will stay powered. It must be taken into consideration that no parasitic currents will flow via signal lines between inactive and active sections.

The individual areas are explained below.

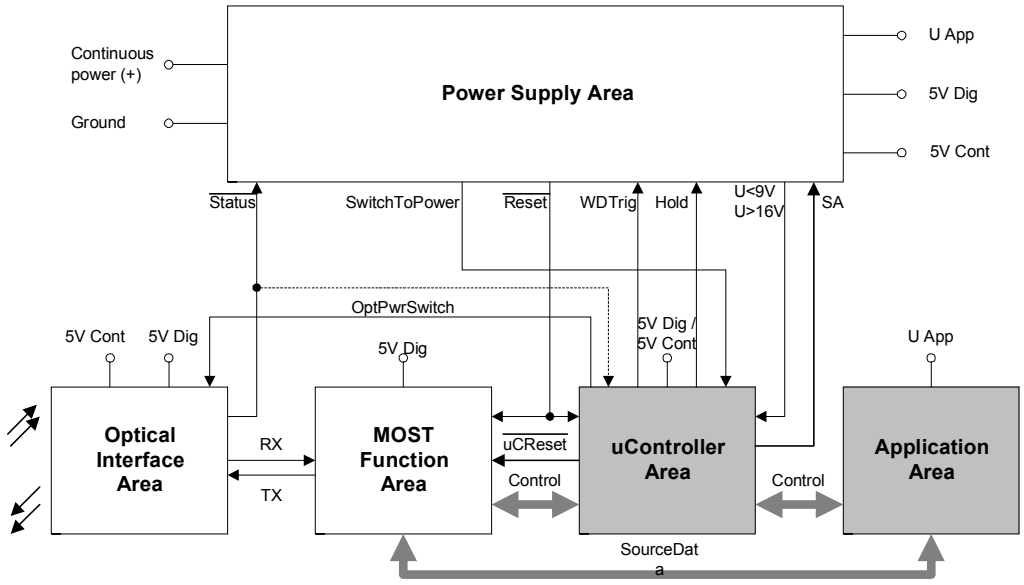


Figure 4-1: Structure of a MOST Device; The different functional areas and their interfaces.

4.2 Optical Interface Area

4.2.1 Overview

The optical interface area consists of an optical receiver and an optical transmitter. Both communicate with the MOST Transceiver via a single data line (RX and TX). The receiver is connected to continuous power, unlike the transmitter.

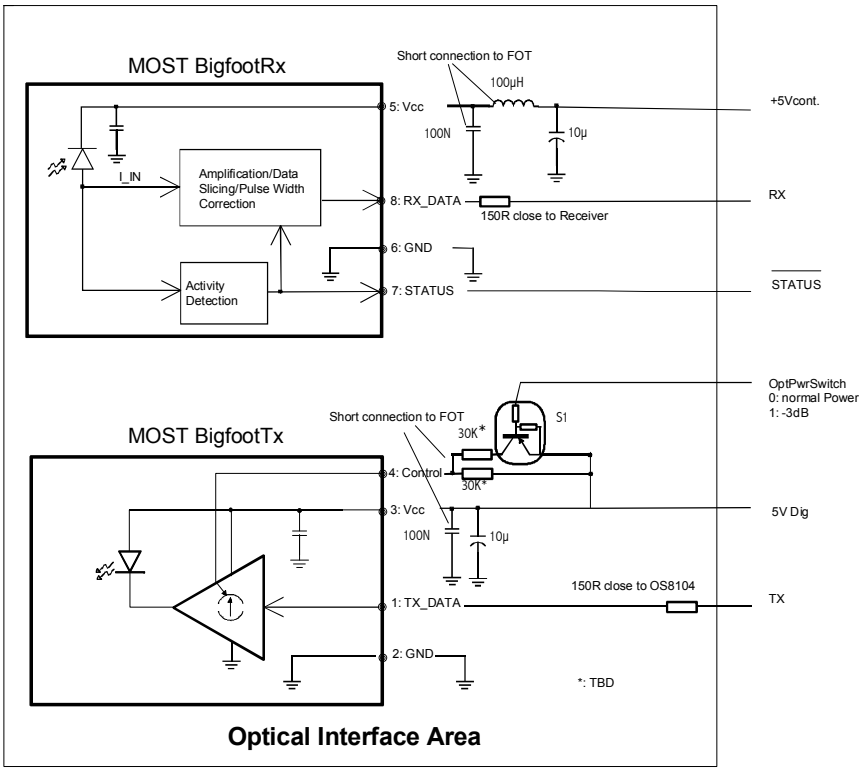


Figure 4-2: Optical interface area.

The receiver contains an ActivityDetection logic that is supplied with continuous power via the micro power regulator (5V cont.), and that consumes less than 20µA. As soon as the ActivityDetection logic recognizes modulated light, signal status is switched to low, and the received data stream is switched to the output.

Signal **Status** is connected to the power supply area, and therefore the power to the transceiver, and eventually to other areas, is switched on.

If no light is received, the receiver is switched off, except for the wake-up logic. Signal **Status** is high then.

It is possible to influence the driver current of the transmitter by a resistor between 5Vdig and input Control. A second resistor that has the same value, can be connected in parallel to the first resistor by using switch 1. When doing that, the optical output power is increased by approx. 3dB. Switch 1 is controlled by signal **OptPwrSwitch**, which is driven by the µC area. The µC only activates **OptPwrSwitch**, if it has received the respective KWP2000 command. After $t_{OptPwrLow}$, the µC has to deactivate OptPwrSwitch independently.

In normal operation mode the switch is closed, so the transmitter runs at maximum optical power output.

This circuit provides diagnosis. If the optical power between two devices is reduced by 3dB and the system still works correctly, it can be assumed that there is a reserve of at minimum 3dB with respect to the optical power budget.

Please note:
By an appropriate arrangement (e.g. Pull down resistor, or inverter connected to OptPwrSwitch) it must be made sure, that S1 is closed in case of an inactive µC.

For the lock behavior in a MOST network there are two important influencing variables:

- Optical Power Budget
- Phase jitter

In opposite to the power budget, the phase jitter may be accumulated when passing several nodes. An important influence variable for the phase jitter is the design of the optical interface area. Here interference's on highly resistive data wiring and crosstalk may occur.

For avoiding that, in the data lines to and from the MOST Transceiver a resistor of 100Ohms up to 150Ohms must be inserted. The resistors must be placed as close to the feeding output as possible. In addition to that, optical receiver and transmitter must be placed closely to the MOST Transceiver The maximum length of data lines to Rx and from Tx must be less than 1.5cm.

Another important factor is the layout of the PCB. Below all data lines, transmitter and receiver a HF ground plane should be placed. Ideally, Rx and TX line are placed as far apart as possible, separated by a piece of ground area. The shielding box of the optical header must be connected well to the ground plane (by soldering). In addition to that, the power supply of the optical interface area must be buffered and blocked carefully. Therefore the bypass capacitors must be placed as close to the transmitter and receiver as possible. 100nF (Ceramic type) must be placed between every VCC and GND. Another important point is, that the bypass capacitors of transmitter and receiver must be located between the transmitter/ receiver and that point, where the two ground planes of receiver and transmitter are connected together.

Please note
Since very high data rates are transported at low signal levels, Optical Interface Area and MOST Function Area must be designed with respect to the rules of high-frequency engineering.

4.2.2 Optical Power Budget

The manufacturer of the two FOT units guarantees, that the optical power specified for the transmitter's side is emitted by the Pig Tail at the socket of the device. In addition to that he guarantees that sensitivity and dynamic of the receiver are within the specified range. The required optical power budget between sender and receiver is calculated below.

The power of the sending diode refers to the optical power after coupling in, and after 1 meter of fiber. The losses caused by coupling in need not be handled separately. The specified received power is the optical power that will actually be received by the receiving diode. Therefore no additional losses caused by decoupling need to be taken into consideration. The aging effects of the diodes and the resulting increasing attenuation are already included in the specified power values.

Worst case:

	Power (minimum)	Losses (maximum)
Minimum transmitting power in the fiber	dBm	
Tolerance of plugs for coupling in, in fiber		------
Plug for coupler plug at device		2 dB
Fiber (10m with 0,3dB/m)		3 dB
Point of separation in door		3.5 dB
Repair		2 dB
Additional losses (By manufacturing of fiber, Assembly, Aging of fiber...)		4 dB
Plug for coupler plug at device		2 dB
Tolerance of plugs for coupling out of fiber		------
Aging and temperature of diode		------
Minimum received power	dBm	
Difference of power/ Dynamics	*dB*	
Sum of losses		*16.5 dB*
Difference	***dB***	

Table 4-1: Optical power budget in worst case, to be required between a pair of FOT units

Every pair of optical transceivers has to realize an optical power budget of at least 16.5dB, related to the power in the fiber under all conditions like variations between different examples of FOTs, aging, temperature, etc.

4.2.3 POF

Connections in the network are made by using plastic optical fiber (POF), resulting in low cost and reduced EMI problems.

4.2.4 Connection Systems (Pig Tail)

In contrast to existing systems, both transceivers are placed remotely with respect to the device's plug, e.g., on the PCB near by the MOST Transceiver. They are connected to the device's socket with Pig Tails (Pieces of POF). This provides the following advantages:

- Possibility of protecting the end of the fiber at the device's socket.
- Easing of EMI problems.
- Flexible placement on PCB.
- Small dimensions.
- Decoupling of the plugging system of the device from the case of the FOTs.

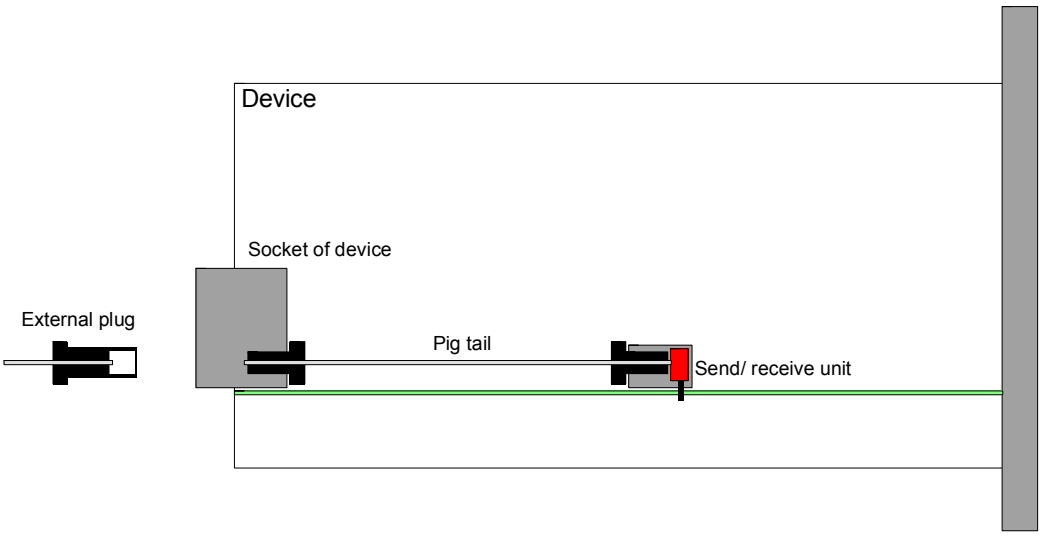


Figure 4-3: Connecting the FOTs to the plug of the device via "Pig tail"

The pig tails are connected to the FOTs and to the device's plug with a plugging system in each case.

The device's plugging system is carried out modularly and in a hybrid way. So one or more pairs of optical connectors can be combined with different electrical connectors as in a kind of model kit, for deriving plugging systems for the different devices.

Please note:
Th description above is one possible implementation of a Pig Tail. More detailed information is to be found in the MOST Physical Layer Specification.

4.3 MOST Function Area

The MOST function area consists of the following components (For more detailed information please refer to [7]):

- MOST Transceiver
- Crystal
- PLL-Filter

In standalone mode, the MOST Transceiver can communicate with I^2C components (as I^2C Master). The transceiver itself is controlled by remote commands via the MOST network. Otherwise it communicates with a micro controller via I^2C (as slave), SPI or parallel bus. Source data is exchanged with the application via the source data bus.

Reset:
On a reset, the MOST Transceiver activates all bypass mode, switches to Slave mode, and switches the interrupt pin to inactive ('1'). After reset is deactivated, the interrupt pin changes to '0' (PowerOn interrupt). The micro controller (µC) waits for this interrupt and then initializes the transceiver in Timing Master or Slave mode. In standalone mode, the transceiver is in all bypass mode during reset. Afterwards it auto-configures itself as Slave.

For devices with µC area it should be possible in any case, that the MOST chip can be reset by the respective µC as well (µC reset or Watchdog Reset), since here the MOST chip is not controlled and initialized via the network, but by the µC.

Please note:
During Reset (not software reset), the signal RMCK is not valid. If RMCK is used as device clock, this must be taken into consideration.

4.4 µC Area

The micro controller (µC) area mainly consists of the µC and some memory, and is not mandatory for a MOST Device (standalone mode). In the case of devices with a µC area, there might be applications that are tightly coupled to the network activity. They need to realize a low standby-current I_{STBY}, so in PowerOff mode of the network, the µC must be switched off.

At the same time there are devices which must be active even if there is no network activity. Here the µC area must be connected to a continuous power supply.

In addition to that, there are devices which are to be arranged in between. They are active without network activity, but are not connected to continuous power (for example, the power supply of a CD changer during eject of disc).

4.5 Application Area

Application area refers to the application peripherals such as receivers, amplifiers, drives, etc. The way of implementing an application area is very device-specific. In some devices, especially those with application peripherals that have high power consumption, it makes sense to supply the peripherals separately from the logic, i.e., the µC area and the MOST function area, in order to switch them on and off separately. In other applications, the application area must be connected to a continuous power supply.

If internal communication is required, the MOST network with all devices connected to it is powered. Since this may happen also in such cases where the vehicle is parked, the power consumption in this communication mode (Logic_Only_Mode) must be kept as low as possible (not only in sleep mode). This means, that it must be possible to remove the Application Area from power (if procurable).

4.6 Power Supply Area

Please note:
The voltage levels shown in this section here could vary between the systems. Therefore, they are non-normative. Binding values must be defined in the specifications of the System Integrators.
"Power Supply Area" describes the power supply for a device that is usually active when the network is active, so a low standby-current I_{STBY} must be achieved. This is the most complex case. Figure 4-4 shows an example for the implementation of a Power Supply Area.

To meet these requirements, a MOST Transceiver, micro controller (µC), and application peripherals are completely separated from power. In addition to that, the application periphery is powered separately, so that it can be switched off although the logic is still running (e.g., drive).

The implementation of the power supply area, as shown in Figure 4-4, mainly consists of:

- Filter, unload-protection, EMI/EMC protection
- Micropower regulator (5V Cont.)
- SwitchToPower detector
- Power on logic
- Digital power supply (5V Dig)
- Application power supply (U App)
- Bad power condition comparator
- Reset generator
- Watchdog timer

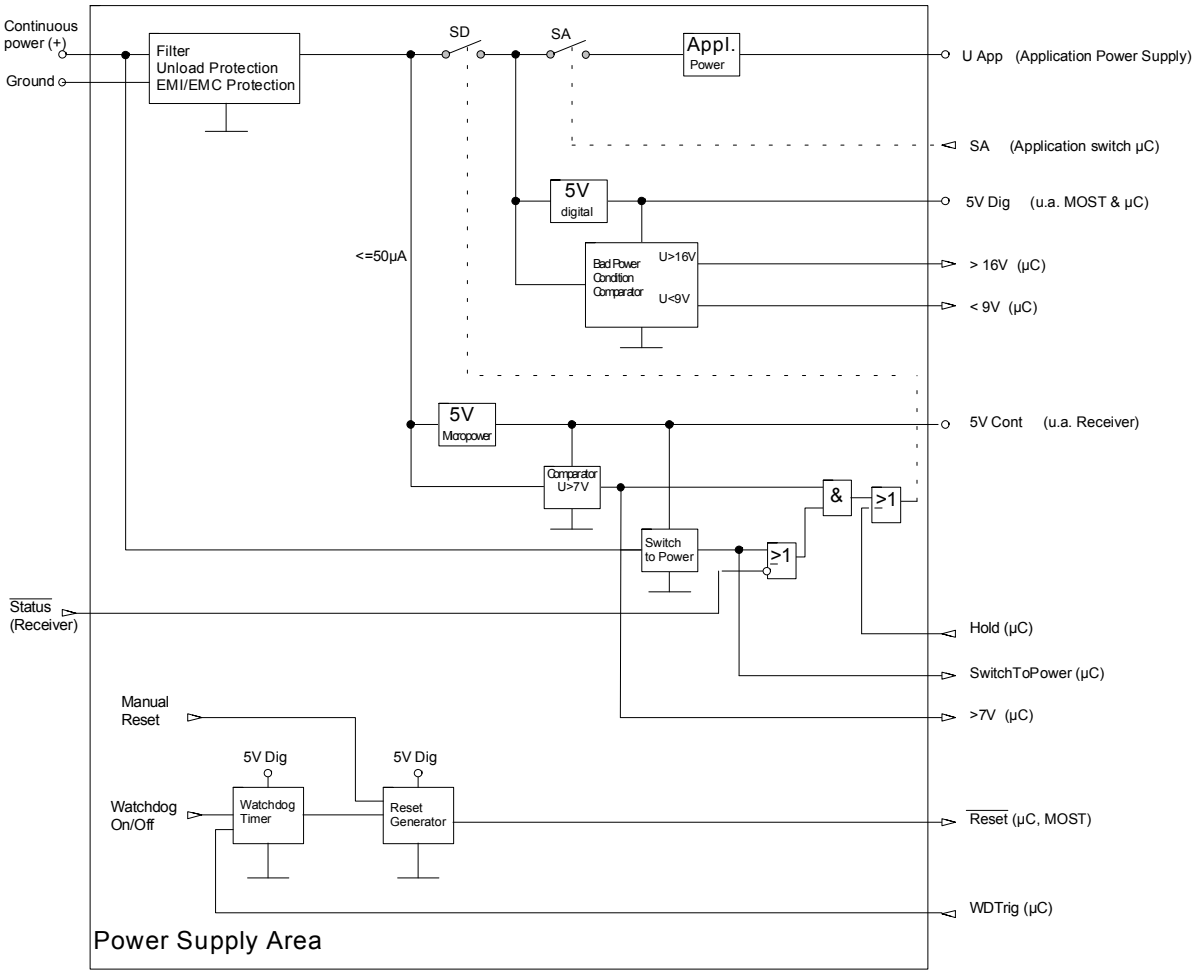


Figure 4-4: Block diagram of power supply area.

Filter and **EMI/EMC-Protection** filters the power supply and protects the device from incoming radiation, or it prevents the device from sending out radiation. The **Unload-Protection** provides the overcoming of short periods of low voltage.

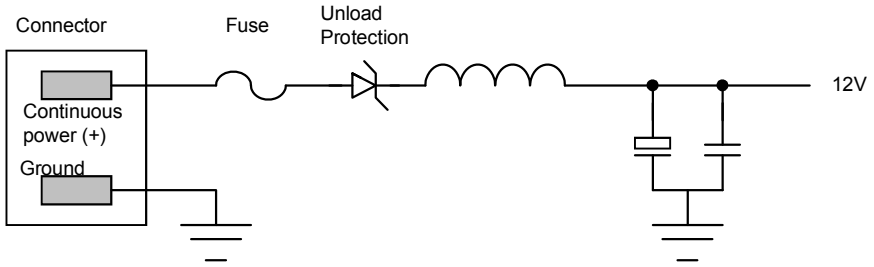


Figure 4-5: Input section of power supply area.

The **Micropower Regulator** provides power supply for the receiving FOT unit with wake-up functionality (Please refer to [6]), and of the Power On logic, if the device is switched off on an inactive bus. Furthermore, it can be used to supply volatile memory devices. In total, the device has to meet a manufacturer-specific standby current I_{STBY}. In case of the devices that stay active on an inactive network, or that become active from time to time on an inactive network, the **Micropower Regulator** must be dimensioned to provide more power.

The **SwitchToPower Detector** is used for ring break diagnosis, where the location of an interruption of the ring is localized. This is not done during normal operation, but in the car repair, or at the assembly line. Since the bus cannot work properly on a ring break, the devices must get a trigger in another way. Such a trigger is set by a defined switching off of the power supply of all devices for some seconds by a central power switch. The switched-off state should be maintained for some seconds, because all devices should be completely unloaded.

Ring break diagnosis is started by switching on power by the central power switch. The SwitchToPower detector recognizes that the device powered up, and generates a pulse, by which power of the device stays activated for a certain time. After the reset phase, the micro controller (µC) recognizes with the help of the SwitchToPower signal that the device was powered, and switches to ring break diagnosis mode. Before this, Hold must be activated to prevent the device from being switched off again.

If no communication is started on the network, the µC must deactivate Hold so the device can switch back to sleep mode.

The SwitchToPower detector must be implemented so that the SwitchToPower pulse is generated only if the power sinks below a certain threshold. Under no circumstances should short breakdowns on the supply voltage (e.g., by the starter) lead to a SwitchToPower pulse.

Therefore the SwitchToPower detector gets armed only, if the device was separated from power for at least 2 seconds, and at most 4 seconds (2sec < t1 < 4sec). Only then will it generate a pulse when the device is connected to power. This must be made sure of with the help of suitable measures (unload protection diode, and individual electrolyte capacitor at the power supply line of the detector).

In addition to that, the SwitchToPower detector must supervise the power supply before unload protection, since, caused by the switching off of all function areas, the voltage will decrease very slow.

The SwitchToPower pulse must have a minimum length t2, which must be long enough for the µC to safely recognize the pulse.

This is shown in the figure below for ideal signals (vertical edges). The SwitchToPower detector should be implemented at low cost, and in a way so that it works on ideal conditions as shown in the figure below. It should not be tried (at high cost) to meet the timing exactly, even on non-ideal conditions, since imprecise behavior of devices can be compensated for by the duration of the real trigger, and it is not very critical if, on an alleged trigger (e.g., caused by starting the engine), a device inadvertently switches to diagnosis mode.

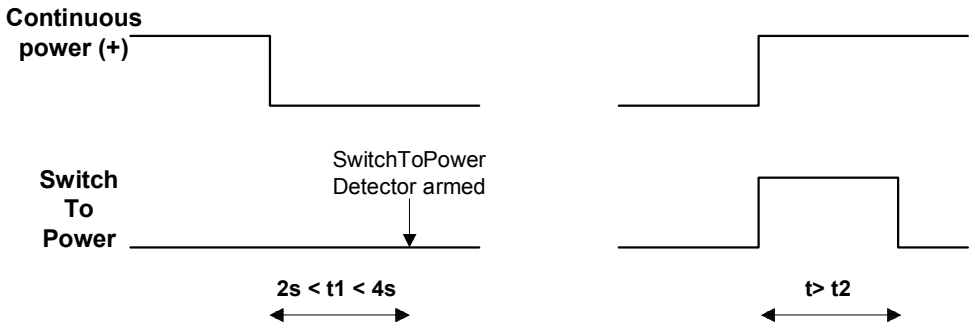


Figure 4-6: Timing of the output signal of the SwitchToPower detector depending on voltage at continuous power input.

The **Power On Logic** checks to see whether the bus is active, or if the SwitchToPower detector indicates that the device is freshly connected to power. If, in addition to that, the U>7V comparator indicates a sufficient supply voltage, switch SD is closed and **Digital Power Supply** is connected to power. **Digital Power Supply** then supplies the MOST Transceiver and the micro controller (µC). As soon as the µC is started, it keeps switch SD closed by an additional input to the Power On Logic (Hold).

Later on, the µC decides if and when the application periphery (application area) will be powered, and activates SA.

The **7V comparator** indicates whether the input voltage is less than 7V (U_{Low}) or not. It is important to implement a hysteresis here, since when switching off the supply voltage due to low voltage, the voltage at the input of the comparator will suddenly be increased again. Without hysteresis, the device would be switched on again, leading to an oscillation of the 7V comparator, and of the entire digital supply voltage.

Please note:
The hysteresis must be implemented in a way, that the output signal of the 7V comparator is switched off, when the voltage drops below 7V. The output signal of the 7V comparator must then be switched on again only, if the voltage rises to >9V (+1V/ -0.5V).

The 7V comparator must behave in a defined way, even if the voltage keeps decreasing and the micro power regulator does not stabilize its output voltage, but follows the input voltage only. That means that the 7V comparator must prevent the device from switching on down to a voltage level <2V..3V.

The **Bad Power Condition comparator** recognizes critical voltage ($U_{Critical}$; U<9V) and super voltage (U>16V) on the supply power, so that appropriate actions can be taken. For the Bad Power Condition signals, a hysteresis is not mandatory, since they do not control switching off of power. The signals are evaluated only by the micro controller (µC).

The **Reset Generator** generates reset for the MOST Transceiver and eventually for a µC if available. It is mandatory for all devices! Possible sources for reset are:

- Device connected to power
- Transition between low voltage to normal operation
- Low voltage on power
- Manual reset (reset button)
- Watchdog timer

The maximum length of the reset pulse is 300ms.

If a µC is available, a **Watchdog Timer** (eventually with an integrated reset generator) is mandatory. The watchdog timer initiates a reset at the reset generator, when not triggered by the µC for a certain time (WDTrig). This closes the All Bypass of the MOST Transceiver. Even if the application processor does not restart, the device behaves in a neutral manner with respect to the bus. If a device has no µC, no watchdog timer is required.

Please note:
It must be made sure, that the HOLD mechanism (by which the µC keeps the device powered) is reset as well. The MOST Transceiver can be reset by the µC as well.

4.7 Voltage Levels

In general, a device in sleep mode must not wake the bus (light on), caused by low voltage or super voltage. Four voltage ranges are defined:

Normal operation: **[9V < U < 16V]**

Device works normally, all functions are within the specified tolerances.

Super voltage: **[U > 16V(+/- 1V)]**

The device is in a safe operation state, which must be defined for each device individually.

Critical voltage ($U_{Critical}$): **[7V(+/- 0.5V) < U < 9V(+1/- 0.5V)]**

The device is in a safe operation state, which must be defined for each device individually. The NetInterface works normally, the device can communicate. On a recovery from this state, the network does not need to be initialized again.

Low voltage (U_{Low}): **[U < 7 V]**

The device is in a safe operation state, which must be defined for each device individually. The voltage has dropped to a value where the device cannot communicate for long. The NetInterface does not work any longer, so a device that can not communicate safely has to switch off light in a safe way.

A safe operation state means that the device must take measures for avoiding failure, overheating, or destruction of its own or connected functional sections. In addition, it must switch to a state from which it can resume working normally if normal voltage is restored.

Please note:
The voltage levels in brackets are non-normative and could vary between different systems.

Examples:

- Muting and eventually switching off of amplifiers (danger of overheating, protection of loudspeakers when switching off caused by low voltage).
- Switching off the servo units of CD/MD player (protecting the optical PickUp).

Remarks:

1. The specified voltage thresholds are minimum values. This means that the application must be able to work between 9V and 16V, and that the critical voltage area reaches 7V. This does not mean that it should not been tried e.g. to enter Low Voltage at lower voltages if this is possible. Especially when using switched power supplies, it can be possible to drop the Low Voltage threshold to e.g. 3V.

2. The hysteresis ranges must be implemented for avoiding oscillating!

Low Voltage for a short period of time:

Some devices need a long time for initialization (Operating system, system communication...). If such a device would be reset even at short pulses of low voltage, it needed to be initialized after that. The interruption that would occur with respect to the entire system, would be recognizable by the customer (e.g. interruption of audio when starting the engine). Such a device should be able to survive short Low Voltage periods, without the need of being re-initialized. Especially the initialization status of the µC must be secured. This might be done, e.g., by using buffer capacitors, unload protection diode, a separate power supply for the digital section, releasing of the application peripherals, stopping the µC, etc.. Also the operation of the NetInterface can be reduced, e.g. by resetting the MOST Transceiver, which will then close its All Bypass (Except a device containing the Timing Master). The light should be kept switched on as long as possible, since then the rest of the system would not be disturbed. After the Low Voltage period the MOST Transceiver will be re-initialized (in total < 100ms). For more information about the behavior of the software in case of Low Voltage please refer to section 3.2.6.6 on page 141.

5 Tools

5.1 OptoLyzer*4*MOST®

OptoLyzer*4*MOST® of Oasis SiliconSystems is an optical analysis tool for the MOST network. It provides the following features:

1. **Viewer:**
   It is possible to open several viewer windows, to conveniently analyze the data traffic on the control channel of the bus, either online or offline. Messages are automatically disassembled based on the syntax tree.
2. **Syntax Tree:**
   Telegrams for controlling devices can be assembled (sending) or disassembled (receiving) by a syntax tree. The syntax tree is user definable (text based) and is compiled via the syntax compiler.
3. **Filter:**
   Relevant information can be extracted based on a variety of possible adjustments.
4. **Recorder:**
   Traffic on the control channel can be recorded and saved.
5. **Sender:**
   Control messages can be sent via the bus. They can be assembled based on the syntax tree.
6. **Remote Control:**
   Panels for remote controlling of devices can be generated, which can simplify repetitive operations.
7. **Timing Analysis:**
   Analysis of traffic and response times of devices on the bus.
8. **Source Data Control:**
   Various possibilities for influencing channels in the synchronous area.
9. **Synchronous Interface:**
   Several analog and digital I/Os for synchronous signals at the OptoLyzer*4*MOST® Interface Box.
10. **Stress Functions:**
    Bus load generation and error simulation on the MOST network.
11. **Spy Mode:**
    Viewing the entire traffic on the control channel of the MOST bus by an OptoLyzer*4*MOST® Interface Box that is “invisible” for devices at the bus.

The OptoLyzer*4*MOST® system consists of hardware (OptoLyzer*4*MOST® Interface Box), which is connected via RS232 to a PC, and PC software, which provides an interface for other applications.

5.2 MOST RapidControl

The software tool MOST RapidControl is based on the OptoLyzer*4*MOST® PC software. It provides fast and effective generation of operating and display panels, which can be individually designed to meet a wide range of needs. It is possible to assign commands to operation elements to be activated on certain operations. Display elements can filter the relevant telegrams out of the data stream. In addition to that, functions can be defined to translate the sequence of bits in the telegrams to a selected display format. The syntax tree of the OptoLyzer*4*MOST® PC software can also be used.

5.3 OptoLyzer*4*MOST® Professional

5.3.1 Introduction

5.3.2 Architecture

The basis of OptoLyzer*4*MOST® Professional is the OptoLyzer® Socket Library. It consists of a collection of library functions which cover all aspects of analysis and simulation of a MOST network. OptoLyzer® Socket Library has a generic interface for the hardware. Adapting to different OptoLyzer hardware components is done by device-specific libraries (DLLs), which can be integrated into the system as needed, e.g., together with new hardware.

Device-specific DLLs are provided for connecting the OptoLyzer*4*MOST® Interface Box via RS232 and for PC-Boards via ISA/ PCI bus. The structure of the OptoLyzer® Socket Library is based on three main functional blocks:

- OptoLyzer control block
- Bus analyzer
- Simulation interface

5.3.3 OptoLyzer Control

The OptoLyzer control block connects the OptoLyzer*4*MOST® hardware to the OptoLyzer control software. This block administers global settings such as the usage of PC interfaces. In addition to that, the specific properties of the respective OptoLyzer*4*MOST® Interface Boxes are mapped, e.g., Codec, signal processing, mapping of PC sound channels, Wake-Up, etc.

These properties can be visualized with control windows, which are partly integrated in the basic software. Other control windows are based on independent executables (control modules), which can be added to the system later.

Control modules available to the system are automatically found and integrated in the task bar, so that the user sees one integrated system.

Programs listed below are examples of plug-ins which can be integrated into the system:

- MOST Edit, an easy-to-use register editor for the MOST Transceiver OS8104.
- MOST Radar, an analysis tool for exploring every MOST network in detail.
- Oasis Flashwriter, a tool for automatic firmware updates for OptoLyzer*4*MOST® Interface Boxes and all connected MOST Devices.

New hardware developed for MOST can be integrated with the system by providing a system-specific DLL and a new control window. Since the system is very flexible, it can accommodate new abilities of the hardware as well.

5.3.4 Bus Analysis

This function block collects bus events, such as control messages, lock, unlock, light, or error conditions. All events are provided with a time stamp and collected in a central message buffer. Bus analysis provides several functions with a uniform interface to the visualization software, including filtering functions, functions for recording events, and functions for generating trigger conditions.

The analysis windows for viewing data are also based on the OptoLyzer® Socket Library. By default, the "classic" analysis windows like Viewer, Timing Analyzer and Recorder are built in. In addition to that, modules can be added later for more detailed analysis.

This makes it possible to generate complex recording modules with pre- and post-triggering, or specific viewers including special disassembling functions. A very interesting application would be to add recording modules that evaluate and record events on the MOST network and special data at the same time. This way it would be possible, e.g., to record the audio output of a tuner together with the respective messages and events on the MOST network. Of course, the hardware environment must meet the requirements of this application.

Since the bus analysis function block is closely associated with the simulation interface, messages from the virtual MOST network can also be recorded.

5.3.5 MOST Simulation Interface

When developing applications for MOST Devices, simulating the application in a virtual MOST environment helps to optimize the development process. A simulation can be used for testing before the actual hardware is available. The optimal approach for simulation is to combine simulation of virtual MOST Devices with real-world devices. It is implemented in this way in OptoLyzer*4*MOST® Professional.

OptoLyzer*4*MOST® Professional consists of a collection of library functions covering all aspects of analysis and simulation of a MOST network. OptoLyzer® Socket Library has a generic interface for the hardware. Adapting to different OptoLyzer hardware components is done by device-specific libraries (DLLs), which can be integrated into the system subsequently (e.g., together with new hardware).

An application written in ANSI-C can be simulated just by adding a few functions that handle connection to the OptoLyzer® Socket Library. When implementing the simulated application into hardware, the simulation-relevant functions can be disabled by removing a single switch.

The simulation function of OptoLyzer*4*MOST® Professional covers all steps of development of a MOST system. It is possible to simulate virtual MOST Devices communicating with other virtual MOST Devices in a virtual MOST network. In addition to that, it is possible to mix real and virtual MOST Devices, so that virtual MOST Devices can communicate with real devices.

It is the simulation interface that links the virtual and the real MOST networks. It provides a message interface, which can support almost any number of virtual MOST Devices. The interface itself consists of functions that provide sending and receiving of MOST messages and the handling of group address and logical address. For a simulated MOST Device, the OptoLyzer® Socket Library behaves like a real MOST node. In addition to that, a real MOST Transceiver can be assigned to a virtual MOST Device in an OptoLyzer*4*MOST® Interface Box or a PC Board. The simulated MOST Devices become "visible" and addressable for real MOST Devices.

The simulation interface functions are equivalent to the functions of the basic layer (Layer I) of the MOST NetServices API, which are a library implemented in ANSI C providing software access to the MOST network.

This ability to simulate virtual devices in a virtual network based on the OptoLyzer® Socket Library allows the interaction between MOST Devices to be tested before they have been developed. This is a very useful feature when defining new systems.

With the OptoLyzer® Socket Library, several PC boards or OptoLyzer*4*MOST® Interface Boxes can be connected to cover MOST systems of any complexity. Since the OptoLyzer® Socket Library offers a variety of interfaces, it is no problem to find an appropriate development platform.

6 Appendix A: Index of Figures

7 Appendix B: Index of Tables

8 Appendix C: INDEX

µ

7

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Notes:

Notes:

Annex 3

MOST

Media **O**riented **S**ystems **T**ransport

Multimedia and Control Networking Technology

MOST Specification Of Physical Layer
Rev 1.0
02/2001
Version 1.0-00

Legal Notice

COPYRIGHT

LICENSE DISCLAIMER

Nothing on any MOST Cooperation Web Site, or in any MOST Cooperation document, shall be construed as conferring any license under any of the MOST Cooperation or its members or any third party's intellectual property rights, whether by estoppel, implication, or otherwise.

CONTENT AND LIABILITY DISCLAIMER

MOST Cooperation or its members shall not be responsible for any errors or omissions contained at any MOST Cooperation Web Site, or in any MOST Cooperation document, and reserves the right to make changes without notice. Accordingly, all MOST Cooperation and third party information is provided "AS IS". In addition, MOST Cooperation or its members are not responsible for the content of any other Web Site linked to any MOST Cooperation Web Site. Links are provided as Internet navigation tools only.

MOST COOPERATION AND ITS MEMBERS DISCLAIM ALL WARRANTIES WITH REGARD TO THE INFORMATION (INCLUDING ANY SOFTWARE) PROVIDED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

In no event shall MOST Cooperation or its members be liable for any damages whatsoever, and in particular MOST Cooperation or its members shall not be liable for special, indirect, consequential, or incidental damages, or damages for lost profits, loss of revenue, or loss of use, arising out of or related to any MOST Cooperation Web Site, any MOST Cooperation document, or the information contained in it, whether such damages arise in contract, negligence, tort, under statute, in equity, at law or otherwise.

FEEDBACK INFORMATION

Any information provided to MOST Cooperation in connection with any MOST Cooperation Web Site, or any MOST Cooperation document, shall be provided by the submitter and received by MOST Cooperation on a non-confidential basis. MOST Cooperation shall be free to use such information on an unrestricted basis.

TRADEMARKS

MOST Cooperation and its members prohibit the unauthorized use of any of their trademarks. MOST Cooperation specifically prohibits the use of the MOST Cooperation LOGO unless the use is approved by the Steering Committee of MOST Cooperation.

SUPPORT AND FURTHER INFORMATION

For more information on the MOST technology, please contact:

MOST Cooperation
Administration
P. O. Box 4327
D-76028 Karlsruhe
Germany

Tel: (+49) (0) 721 966 50 00
Fax:(+49) (0) 721 966 50 01
E-mail: contact@mostcooperation.com
Web: www.mostcooperation.com

Contents

Bibliography

Number	Document
[1]	MOST Specification Framework
[2]	MOST Specification
[3]	MOST High Protocol Specification
[4]	MOST NetServices “Basic Layer”; User Manual and Specification
[5]	MOST NetServices “Application Socket”; User Manual and Specification
[6]	Obsolete;
[7]	Obsolete;
[8]	MOST FunctionCatalog
[8]	MOST Specification Of Physical Layer

Document History

First version 1.0-00

Change Ref.	Section	Changes
-	-	- First version, no changes

1 Introduction

The Physical Layer of a MOST network supports the optical transmission of the bit stream between MOST Devices. This document describes and defines the "*MOST Specification of Physical Layer*".

The physical connection of two MOST devices is called "*point-to-point-link*". The "*MOST Specification of Physical Layer*" is organized in four specification points along an individual "*point-to-point-link*". The location of the specification points is shown in Figure 1-1.

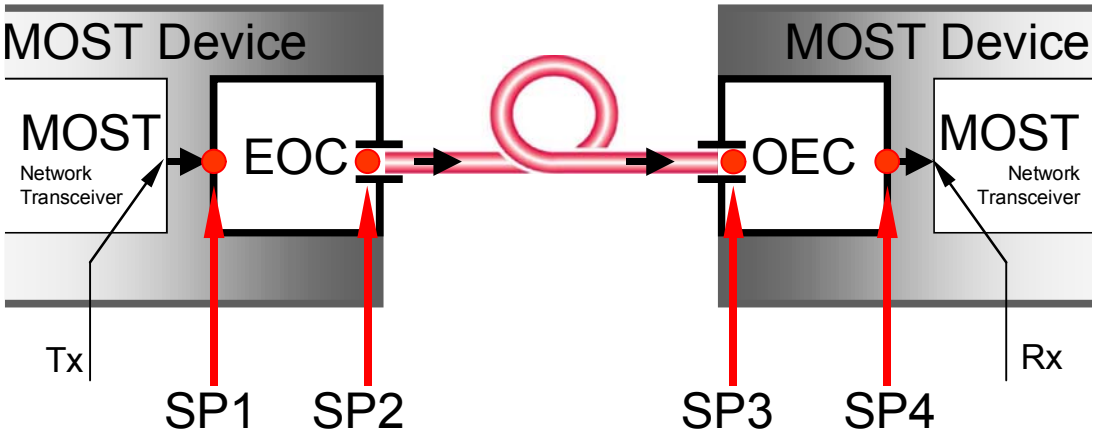


Figure 1-1: Location of specification points along a MOST point-to-point-link.

Specification point No.1 and specification point No. 4 are pure electrical interfaces. They define the electrical signal requirements between a MOST transceiver unit (with its input Rx and output Tx) and an optical ⇔ electrical converter (e.g. electrical signal timings, electrical signal levels, electrical signal distortion).

Specification point No. 2 and specification point No. 3 define the optical and mechanical properties of the interfaces between a MOST device and a wiring harness (e.g. wavelength, optical signal timing, optical power budget, optical signal distortion, connector interface drawings). All parameters and definitions required for individual specification points are summarized in tables that are referenced in Table 1-1.

Individual MOST devices that fulfill the "*MOST Specification of Physical Layer*" are basically compatible to each other. Due to several point-to-point-links within a MOST network ring, system network functionality is only ensured, when "*MOST Specification of Physical Layer*" is fulfilled by every individual point-to-point-link and the special requirements of section 2.6 are considered. The number of MOST devices is equal to the number of point-to-point-links.

There are special tests defined to check the physical layer compliance of MOST devices. These tests are part of the general MOST compliance test (see references in Table 1-1).

	References within this document	References to other documents / URL
SP 1: Electrical signal definitions	Table 2-1	
SP 2: Optical signal definitions Connector interface drawings	Table 2-2 Table 3-1 / Table 3-2 Figure 3-1	www.mostcooperation.com
SP 3: Optical signal definitions Connector interface drawings	Table 2-3 Table 3-1 / Table 3-2 Figure 3-1	www.mostcooperation.com
SP 4: Electrical signal definitions	Table 2-4	
Additional Requirements for System Design	Section 2.6	
Compliance Test of Specification Points		www.mostcooperation.com
Commercial Products and proprietary Specifications		www.mostcooperation.com

Table 1-1: Content and references of MOST Specification of Physical Layer.

2 Electrical and Optical Parameters

2.1 General Remarks

A MOST frame consists of 512 bit. The bit rate B of the MOST system can be calculated as B = 512bit * FS for a given sampling rate (FS = Frame Sync). Due to biphase mark coding the smallest pulse length is given as UI = 1/(2B) with the Unit Interval UI.

Within a MOST pattern, only three different pulse lengths can occur. They are called UI, 2UI, 3UI. In case of FS = 44.1kHz, the Bit Rate B and the Unit Interval UI are given as B = 22.5792 MBit/s and UI = 22.14ns.

The points Tx and Rx represent the interfaces of the used MOST network transceiver. They are not specified here. However, they have to fulfill the requirements of SP1 and SP4. This includes possible worsening due to the transmission between MOST network transceivers and the respective EOC and OEC.

Please note:
All timing parameters in the following tables are specified for high pulses!

The pulse width variation (t_{pwv}) is defined as the sum of the Average Pulse Width Distortion plus high-frequency jitter (see Figure 2-1). The limits for the pulse width variation are valid for all data elements (Length of high pulse = n*UI; n=1,2,3). Absolute pulse width has to be within the limits

$n*UI - (UI-t_{pwv(min)})$
and
$n*UI + (t_{pwv(max)}-UI)$

with an error rate of better than 10^{-9}.

The Average Pulse Width Distortion, defined as $t_{apwd} = (t_{pwmx} + t_{pwmn} - 2UI)/2$, is illustrated in Figure 2-1. The values t_{pwmx} and t_{pwmn} are maximum and minimum **measured** values, so that pulses with a width of less than $t_{pwv(min)}$ or more than $t_{pwv(max)}$ occur with a probability of less than 10^{-9}. The example of Figure 2-1 shows the limit of $t_{pwmx} = t_{pwv(max)}$ and $t_{pwmn} = t_{pwv(min)}$.

The limits for the Average Pulse Width Distortion are valid for all data elements (Length of high pulse = n*UI; n=1,2,3).

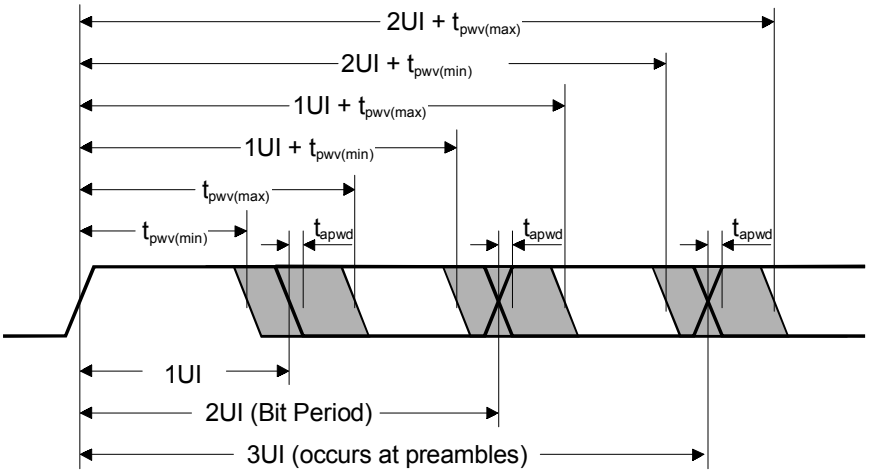


Figure 2-1: Pulse-Width Variation and Average Pulse Width Distortion.

2.2 Specification Point SP1

	Symbol	condition	Min.	typ.	max.	unit
Bit rate		Biphase Mark	4	22.6	25	MBit/s
Low Level Input Voltage	V_{IL1}		-0.3	-	0.8	V
High Level Input Voltage	V_{IH1}	1)	2.0	-	VDD+0.3	V
Rise time (10%-90%)	$t_{r1\ 10\text{-}90}$	2)	-	-	0.23	UI
Fall time (90%-10%)	$t_{f1\ 90\text{-}10}$	2)	-	-	0.23	UI
Pulse Width Variation	t_{1pwv}	3), 4)	0.955	-	1.045	UI
Average Pulse Width Distortion	t_{1apwd}	3), 4)	-0.023	-	+0.023	UI
Input resistance	R_{I1}		2	-	-	kOhm
Input capacitance	C_{I1}		-	-	10	pF

Note:
This table describes the electrical input signal parameters of an electrical optical converter (see figure 1-2). All values have to be guaranteed under automotive worst case conditions (e.g.: Ta = –40°C ... +85°C)

1) Range of VDD = 4.75 V . 5.25 V
2) In case of using 20 % - 80 % measurement $t_{r1\_20\text{-}80}$, $t_{f1\_80\text{-}20}$ has to be 0.18 UI
3) Measured at 1.5 V
4) For more information please refer to 2.1 General Remarks

Table 2-1: Signal parameters of Specification Point 1.

2.3 Specification Point SP2

	symbol	Condition	min.	Typ.	max.	unit
Peak Wavelength	λ_2		630	650	685	nm
FWHM	$\Delta\lambda_2$		-	-	30	nm
Optical output power	P_{opt2}	1), 2)	-10	-	-1.5	dBm
Optical output power "Light off"	P_{OFF2}	2), 3)	-	-	-50	dBm
Extinction ratio	r_{e2}	4)	10	-	-	dB
Rise time (20%-80%)	t_{r2}		-	-	0.27	UI
Fall time (80%-20%)	t_{f2}		-	-	0.27	UI
Pulse Width Variation	t_{2pwv}	5), 6)	0.943	-	1.102	UI
Average Pulse Width Distortion	t_{2apwd}	5), 6)	-0.023	-	+0.068	UI
Positive Overshot within 2/3UI		7)	-20	-	+25	%
Negative Overshot within 2/3UI		7)	-10	-	+20	%
High Level Signal Ripple between 2/3UI and 3/4U		7)	-10	-	+10	%

Note:
This table describes the optical output signal parameters of an electrical optical converter (see figure 1-2). All values have to be guaranteed under automotive worst case conditions (e.g.: Ta = –40°C ... +85°C)

1) Average values, when transmitting modulated light @ MOST-Frame @ signal timing parameters as defined in next chapter
2) Power within a far field angle of 30° (NA = 0.5)
3) Average value, when MOST network off
4) r_{e2}=10*log(b1/b0) average see EN/IEC 61280-2-2
5) Values at 50% of signal amplitude
6) For more information please refer to 2.1 General Remarks
7) The positive/negative overshot and the signal ripple is related to the signal amplitude b1-b0. For more detail refer to Figure 2-2

Table 2-2: Signal parameters of Specification Point 2.

Figure 2-2 shows a schematic diagram of optical pulses.

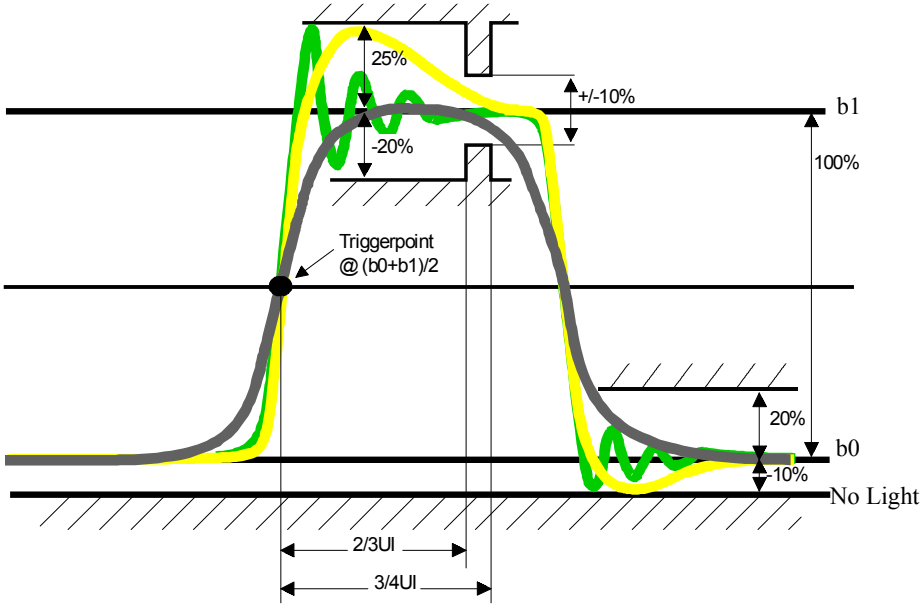


Figure 2-2: Schematic of optical pulses at Specification Point 2.

2.4 Specification Point SP3

	symbol	Condition	min.	typ.	max.	unit
Receivable optical power range for data recovery	P_{opt3}	1), 2), 3), 4)	-24	-	-2	dBm
Receivable optical power range for switching to ”Light off state”	P_{OFF3}	1),2), 5), 3)	-40	-	-24	dBm
Extinction ratio	r_{e3}	6)	10	-	-	dB
Rise time (20%-80%)	t_{r3}		-	-	0.31	UI
Fall time (80%-20%)	t_{f3}		-	-	0.31	UI
Pulse Width Variation	t_{3pwv}	7), 8)	0.943	-	1.102	UI
Average Pulse Width Distortion	t_{3apwd}	7), 8)	-0.023	-	+0.068	UI

Note:
This table describes the optical input signal parameters of an optical electrical converter (see figure 1-2).
All values have to be guaranteed under automotive worst case conditions
(e.g.: Ta = –40°C ... +85°C)

1) Average values, when receiving modulated light @ locked MOST-Frame transmission and no coding errors occur and test pattern which has to be defined, yet.
2) Value, when incoming signal has passed the receiving contact end face
3) Power within a far field angle of 30° (NA = 0.5)
4) Power within a spectral range of SP2
5) Value helpful to create a low-current sleep mode operation, meaning of ”Light off” see MOST specification section 4.2 Optical Interface Area
6) r_{e3}=10*log(b1/b0) average see EN/IEC 61280-2-2
7) Values at 50% of signal amplitude
8) For more information please refer to 2.1 General Remarks

Table 2-3: Signal parameters of Specification Point 3.

2.5 Specification Point SP4

	Symbol	Condition	min.	typ.	max.	unit
Bit rate		Biphase Mark	4	22.6	25	MBit/s
Low Level Output Voltage	V_{OL4}	1)	-	-	0.4	V
High Level Output Voltage	V_{OH4}	1)	2.5	-	-	V
Rise time (10%-90%)	$t_{r4\ 10\text{-}90}$	1), 2)	-	-	0.45	UI
Fall time (90%-10%)	$t_{f4\ 90\text{-}10}$	1), 2)	-	-	0.45	UI
Pulse Width Variation	t_{4pwv}	1), 3), 4)	0.7	-	1.48	UI
Average Pulse Width Distortion	t_{4apwd}	1), 3), 4)	0	-	0.36	UI

Note:
This table describes the electrical output signal parameters of an optical electrical converter (see figure 1-2). All values have to be guaranteed under automotive worst case conditions (e.g.: Ta = –40°C ... +85°C)

1) Assuming a load of 50 kOhm / 10 pF. In case of using 20 % - 80 % measurement $t_{r4\_20\text{-}80}$, $t_{f4\ 80\text{-}20}$ has to be 0.34 UI
3) Measured at 1.5 V
4) For more information please refer to 2.1 General Remarks

Table 2-4: Signal parameters of Specification Point 4.

2.6 Additional Requirements for System Design

The parameters given for the specification points SP1 ... SP4 are the base for a accurate point-to-point link. Looking on a MOST-network with multiple nodes there are additional system-aspects.

The distortion on a MOST-data-stream, described by the parameters Pulse Width Variation and Average Pulse Width Distortion, has to be divided into static and dynamic distortion. All static distortion and parts of the dynamic distortion will be eliminated by an active Most network transceiver chip. The remaining part of the dynamic distortion will be reduced by the transceiver chip and will appear on transmitter-side (SP1). The amount of dynamic distortion on SP1 depends on:

- Dynamic distortion at the input of the transceiver chip (SP4)
- Transfer-function of the transceiver-PLL

The signal quality on SP4 is mainly influenced by the OEC-technology (worst case parameters are given in Table 2-3 and
Table 2-4) and by the optical input power at SP3. The data recovery unit inside the transceiver chip is based on the sampling clock generated by the PLL (Phase Locked Loop) and will attenuate the incoming dynamic distortion.

In a MOST network, the total dynamic distortion based on the Master-TX-signal will accumulate from node to node. Depending on the signal quality at the input of a slave device the total amount of distortion at the output of such a node will increase or decrease.

Therefore, system stability requires a well defined number of links running at high transmission quality to compensate low quality links.

3 Device Connection

3.1 Connector Interfaces

Table 3-1 summarizes the five specified hybrid connector interfaces.

Please note:
The optical contacts located at SP2 and SP3 are combined in a single duplex connector.

"Nick name"	Number of optical contacts [ϕ=1,0 mm]	Number of electrical contacts PIN = 0,63 mm	PIN = 1,5 mm	PIN = 2,8 mm
2+0	2	-	-	-
2+4	2	4	-	-
2+12	2	12	-	-
2+20	2	18	2	-
4+40	2 x 2	2 x 12	-	2 x 8

Table 3-1: Connector family.

Table 3-2 indicates the drawing codes and the file names of the specified connector interfaces.

"nick name"	Drawing code	TIFF File	Drawing Date
2+0	MOST-CON-2-0	MOST-CON-2-0.TIF	6.02.2001
2+4	MOST-CON-2-4	MOST-CON-2-4.TIF	6.02.2001
2+12	MOST-CON-2-12	MOST-CON-2-12.TIF	6.02.2001
2+20	MOST-CON-2-20	MOST-CON-2-20.TIF	6.02.2001
4+40	MOST-CON-4-40	MOST-CON-4-40.TIF	6.02.2001
Fiber Module Interface	MOST-FM-I	MOST-FM-I.TIF	6.02.2001
Test Connector	MOST-CON-T	MOST-CON-T.TIF	6.02.2001
LWL Collar	MOST-FM-C	MOST-FM-C.TIF	6.02.2001
		TIFF Files available on www.mostcooperation.com	

Table 3-2: Drawing codes and file names of connector interfaces.

3.2 Connector Interface Loss

Figure 3-1 shows details of the connector insertion loss D_{con} between devices and cabling:

- The attenuation of optical power may not exceed $D_{2conmax}$ = 2.5 dB, when optical signal transits from SP2 into the cabling fiber.
- The attenuation of optical power may not exceed $D_{3conmax}$ = 2.5 dB, when optical signal transits from the cabling fiber into SP3.

Please note:
The maximum connector attenuation is a part of the "*MOST Specification of Physical Layer*".

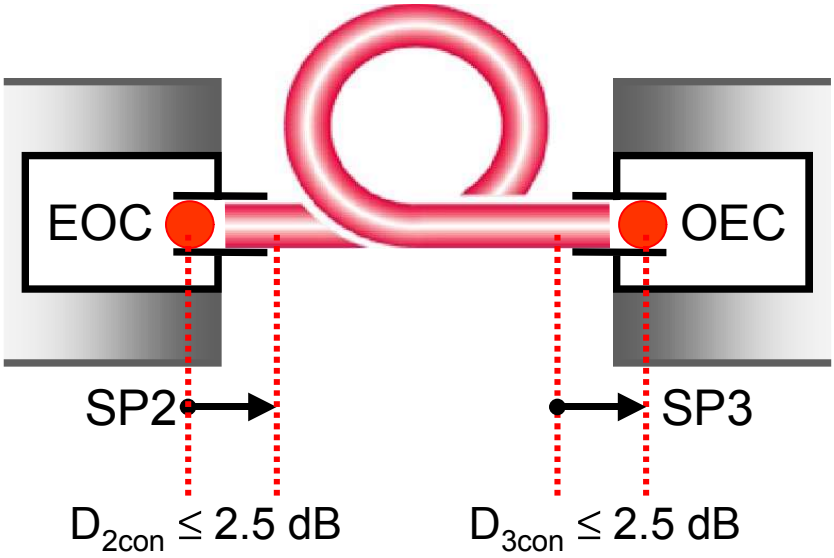


Figure 3-1: Connector Insertion Loss.

4 Appendix A: Index of Figures

5 Appendix B: Index of Tables

6 Appendix C: Abbreviations

APWD	Average Pulse Width Distortion
B	Bit Rate
EOC	Electrical Optical Converter
FS	Frame Sync per second (System sample frequency)
FWHM	Full Width at Half Maximum
NA	Numerical Aperture
OEC	Optical Electrical Converter
PWV	Pulse Width Variation
Rx	MOST Network Transceiver Input
SPn	Specification Point No. n (n= 1 … 4)
Ta	Ambient Temperature
Tx	MOST Network Transceiver Output
UI	Unit Interval

Notes:

Notes:

Annex 4

MOST COOPERATION

TEAM UP WITH MOST. GET ALL.

Downloads II> Specifications II> MOST Framework

SEARCH

Home
Cooperation
Technology
Membership
Downloads
News & Events
Contact

Documents & Downloads

MOST Framework

The objective of the entire specification is to describe the MOST system in terms of physical layer, transport layer, link layer, network management and the programming interface required to develop and build systems and devices that are compliant with this standard. The goal is to provide all information needed to make inter-operable devices in an open architecture but still leaving enough room for product and market differentiation without losing compatibility.

The specification framework is mainly targeted to all those readers, who want to get an introductory overview of the MOST System and its abilities. It provides valuable information to device developers, OEM's and System integrators, but also to independent hardware and software platform architects. The overview can be used for system evaluation, product and system planning.

UP ONE LEVEL

Name	Type	Version	Date	Size	
framework1_1-07.pdf	Adobe PDF	1.1	10/12/2001 12:29:17	663.5kB	DOWNLOAD

MOST Specifcation Framework Rev 1.1 in PDF
Status: November 26, 1999

Annex 5

MOST COOPERATION

TEAM UP WITH MOST. GET ALL.

Downloads II> Specifications

SEARCH

Home
Cooperation
Technology
Membership
Downloads
News & Events
Contact

Documents & Downloads

Specifications

Please read the legal notice:

COPYRIGHT

Copyright MOST Cooperation 1997-2001. All Rights Reserved.

LICENSE DISCLAIMER

Nothing on any MOST Cooperation Web Site shall be construed as conferring any license under any of the MOST Cooperation or its members or any third party's intellectual property rights, whether by estoppel, implication, or otherwise.

CONTENT AND LIABILITY DISCLAIMER

MOST Cooperation or its members shall not be responsible for any errors or omissions contained at this Web Site, and reserves the right to make changes without notice. Accordingly, all MOST Cooperation and third party information is provided "AS IS". In addition, MOST Cooperation or its members are not responsible for the content of any other Web Site linked to any MOST Cooperation Web Site. Links are provided as Internet navigation tools only.

MOST COOPERATION AND ITS MEMBERS DISCLAIM ALL WARRANTIES WITH REGARD TO THE INFORMATION (INCLUDING ANY SOFTWARE) PROVIDED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

In no event shall MOST Cooperation or its members be liable for any damages whatsoever, and in particular MOST Cooperation or its members shall not be liable for special, indirect, consequential, or incidental damages, or damages for lost profits, loss of revenue, or loss of use, arising out of or related to any MOST Cooperation Web Site or the information contained in it, whether such damages arise in contract, negligence, tort, under statute, in equity, at law or otherwise.

FEEDBACK INFORMATION

Any information provided to MOST Cooperation in connection with any MOST Cooperation Web Site shall be provided by the submitter and received by MOST Cooperation on a non-confidential basis. MOST Cooperation shall be free to use such information on an unrestricted basis.

TRADEMARKS

MOST Cooperation and its members prohibit the unauthorized use of any of their trademarks. MOST Cooperation specifically prohibits the use of the MOST Cooperation LOGO unless the use is approved by the Steering Committee of MOST Cooperation.

UP ONE LEVEL

Name	Type	Version	Date	Size	
MOST Framework	Directory				
MOST Framework					
010223_WgPhy_Drawings.zip	ZIP Archive	1.0	10/12/2001 13:29:09	816.1kB	DOWNLOAD
Drawings for MOST Physical Layer Spec. Rev. 1.0					
MOSTHighProtocolSpec_2V1_Doc_2V1-01.pdf	Adobe PDF	2.1	10/12/2001 12:48:32	746.8kB	DOWNLOAD

MOST High Protocol Specification Rev. 2.1

MOSTSpec_Physical_1V0-00.pdf	Adobe PDF	1.0	10/12/2001 12:41:40	572.7kB	DOWNLOAD

MOST Physical Layer Specification Rev. 1.0

MOSTSpecification_2V1_Doc_2V1-00.pdf	Adobe PDF	2.1	11/16/2001 11:04:01	1.6MB	DOWNLOAD

MOST Specifcation Rev 2.1 in PDF (1,5 MB):The MOST Specification (Revision 2.1) contains the detailed specification of the application layer, the network layer, and the MOST hardware.
Status: February 2, 2001

MOST_ContentProtectionScheme_DTCP_1V0-00.pdf	Adobe PDF	1.0	10/12/2001 12:55:00	514.2kB	DOWNLOAD
MOST_ContentSecurity_1V0_00.pdf	Adobe PDF	1.0	10/12/2001 12:52:56	449.2kB	DOWNLOAD

MOST Content Security Specification Rev. 1.0

Annex 6

EE Times Connecting the Global Electronics Community

News & Analysis

Automakers pick MOST as high-speed in-car bus

Charles J. Murray

11/13/2000 05:04 PM EST

Post a comment

Automakers pick MOST as high-speed in-car bus

PARK RIDGE, Ill. — A consortium of the world's biggest automakers last week said it would endorse the high-speed network fiber-optic bus known as MOST (Media Oriented Systems Transfer) by the first quarter of 2001, and possibly as soon as the end of this year. The Automotive Multimedia Interface Collaboration (AMI-C) made the announcement at a technical conference in Turin, Italy.

Endorsement by the AMI-C consortium is viewed as a critical step, because the group includes BMW, DaimlerChrysler, Fiat, Ford, General Motors, Honda, Mitsubishi, Nissan, PSA Peugeot-Citroen, Renault, Toyota and Volkswagen. However, some AMI-C members expect the group will ultimately endorse a second high-speed bus, possibly IEEE 1394.

"No later than first quarter of next year, we should have a steering committee meeting where we endorse MOST," said Michael Noblett, program manager for AMI-C. "Unless something earth-shattering occurs between now and then, it's going to happen."

Use of a common high-speed bus could help carmakers and their suppliers simplify the process of adding to vehicles a host of multimedia devices such as navigation systems, CD players, video screens, digital radios, cell phones and in-car PCs. Today, vendors often redesign products several times to meet the requirements of each automaker's proprietary data bus network.

Royalty-free and open

Endorsement of MOST is seen as a key step for vehicle manufacturers facing an explosion of automotive electronic products. Use of a standard network would let them add new electronics closer to a vehicle's introduction date. For example, manufacturers are already in the planning stage for vehicles that will be introduced in the 2005 model year and later.

For more than a year, automakers have viewed MOST as a candidate. The system, created by the MOST Cooperation (Karlsruhe, Germany), uses a 25-Mbit/second fiber-optic bus. Partners in the Cooperation include Audi, BMW, DaimlerChrysler, Harman/Becker and Oasis Silicon Systems.

Previously, AMI-C members had said that MOST didn't meet the requirements of the consortium's charter, which calls for technologies to be "open and royalty-free." At the Convergence conference in Detroit last month, however, MOST announced that it would eliminate its connection fee of 0.3 Euro per device. The Cooperation's members have also agreed to make all parts of the MOST specification open and available.

"Since they made it open and they made it free, we expect to have no problems with the endorsement," Noblett said.

Several automakers have already incorporated MOST technology into upcoming vehicle programs, typically in luxury cars. BMW will use it in a vehicle to be released next year. Audi and DaimlerChrysler have also said they are integrating the MOST bus into production vehicles.

At last week's ITS World Congress in Turin, other carmakers — including Ford, Jaguar and Volkswagen — demonstrated vehicles containing MOST technology. The technology is also expected to be used by a major American vehicle manufacturer in an upcoming program.

Engineers said that automakers are showing a growing interest in the MOST bus because it offers far higher data rates than buses based on controller-area network (CAN) technology, which is commonly used in today's vehicles for such chores as power-train control. IDB-C (Intelligent Transportation Systems Databus-CAN), for example, offers speeds of about 250 kbits/s, about 100 times slower than MOST. As a result, such CAN-based buses are expected to give way to fiber-optic networks in multimedia applications, although they will continue in their roles under the hood.

Up to now, however, many automakers have maintained allegiance to CAN, even for multimedia, in part because of the lower costs of the copper-based bus. MOST members said they expect that to change, as automotive multimedia applications take off.

"The first few production applications will be luxury vehicles," said Henry Muyshondt, general manager for business development at Oasis Silicon Systems AG (Karlsruhe, Germany). "But it will migrate downward through the vehicle lines as volume builds up."

MOST members also expect the bus to move beyond the European market that developed it. Muyshondt noted out that Ford, Toyota and Nissan are members of the MOST Cooperation.

AMI-C members said the endorsement of MOST would not preclude endorsing another high-speed bus. "At one time, we were going to endorse only a single high-speed network, but that's not the plan anymore," said Edward Nelson, a Ford technical specialist and system team leader for AMI-C.

Room for two

Indeed, most AMI-C members expect IDB-1394, a high-speed bus based on the 1394 computer industry standard, to get the consortium's nod, after some aspects of the spec are brought up to "automotive grade."

Automotive engineers say the two data buses don't necessarily compete with each other and that in some cases, vehicles could combine a 1394 access port with a MOST network.

Many engineers like 1394 because it is compatible with camcorders, video players, DVD systems and other consumer gear that might eventually plug into vehicles. Some engineers are hesitant, however, because unlike MOST, 1394 hasn't been fully proven out in rigorous automotive environments.

Annex 7

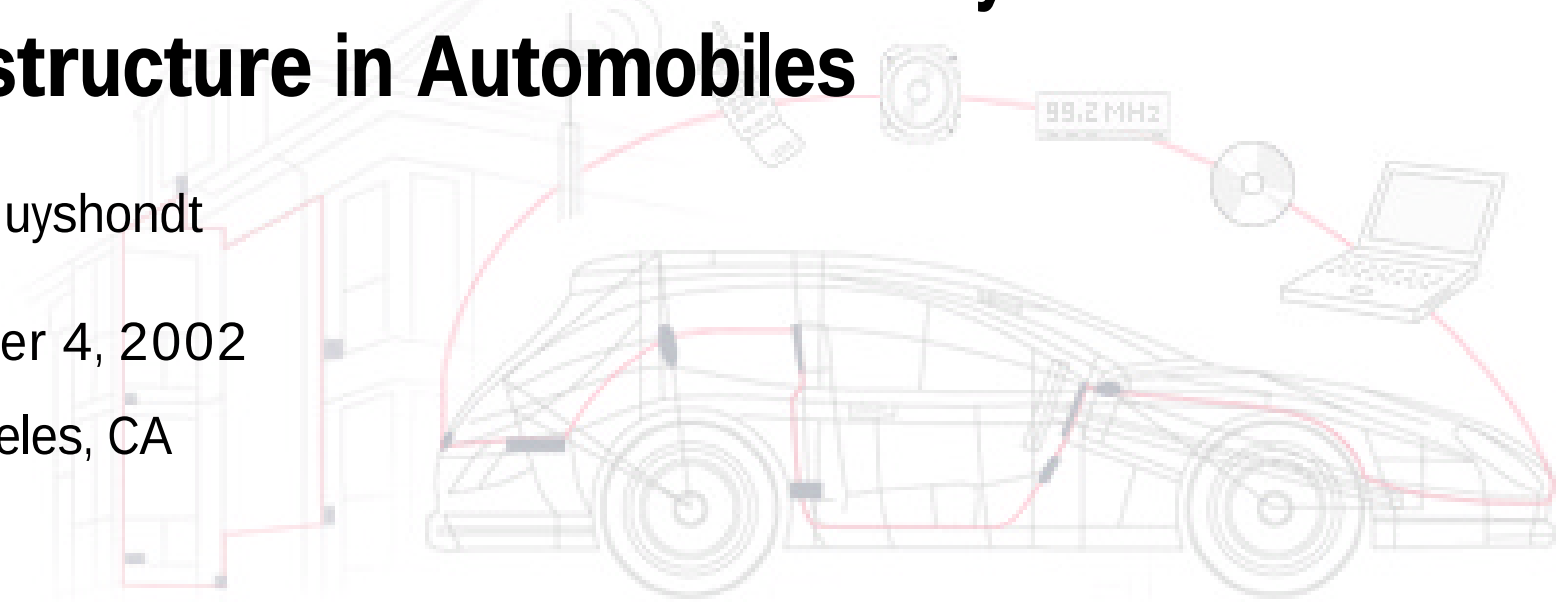
MOST – Media Oriented Systems Transport

Entertainment and Information System Infrastructure in Automobiles

Henry Muyshondt

December 4, 2002

Los Angeles, CA

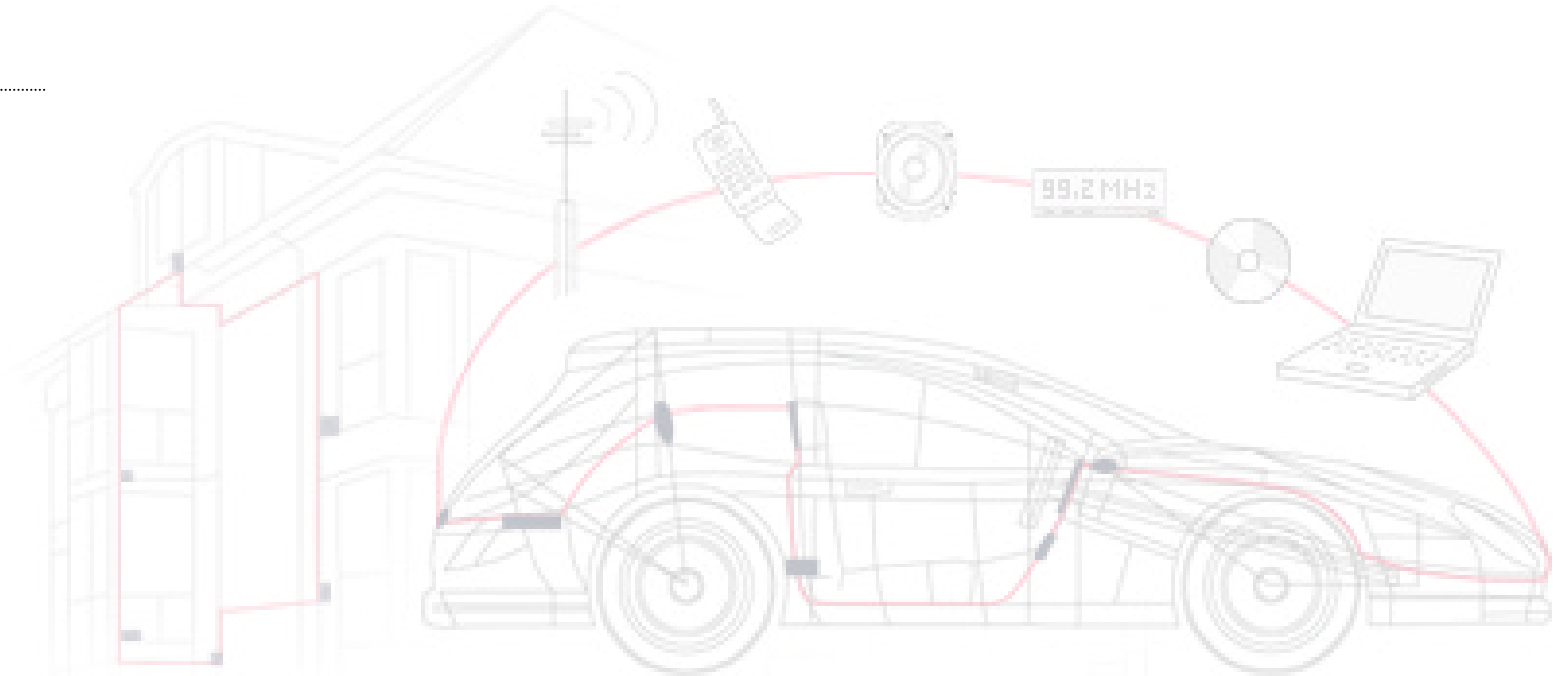


Agenda

Agenda:

- Technology Background
- Adoption in Auto Industry
- Market Outlook
- DRM

– MOST Technology Background
– Adoption of MOST in the Automotive Industry
– MOST Market Outlook
– Content Protection



Technology Background

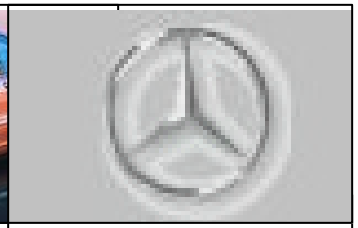
Multimedia Applications Common in Vehicles

Agenda:

- Technology Background
- Adoption in Auto Industry
- Market Outlook
- DRM

- **Multimedia applications moving increasingly into mid-range segments**
- **More components are being installed in each vehicle**
- **Components are increasingly networked with each other**
- **Increased bandwidth used for A / V applications**

Technology Background

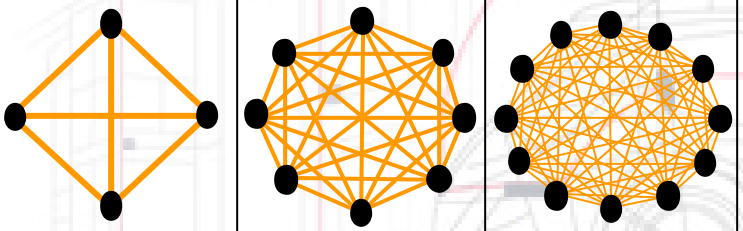
			
1979	**1991**	**1998**	**2002**
1. Radio 2. Amplifier	1. Radio 2. Amplifier 3. CD 4. Telephone	1. Radio 2. Amplifier 3. CD 4. Telephone 5. Microphone 6. Emergency Call 7. Navigation	1. Radio 2. Amplifier 3. CD 4. Telephone 5. Microphone 6. Emergency Call 7. Navigation 8. PDA Interface 9. TV-Tuner 10. DVD 11. Displays 12. Headphones

Increasing Number of Multimedia Components in each Vehicle

- **Simplify interoperation**
- **Develop cost-effective systems**
- **Install what consumers want, when they want it**
- **Increasing entertainment content**

Technology Background

1979	1991	1998	2002
Number of Multimedia Components			
2	4	7	12
Number of Possible Interconnections			
1	6	28	66



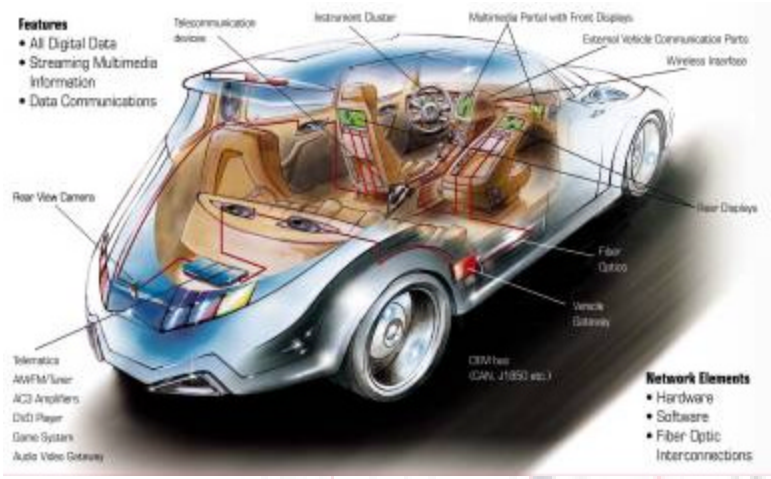
Increasing numbers of Multimedia Components to be connected means:

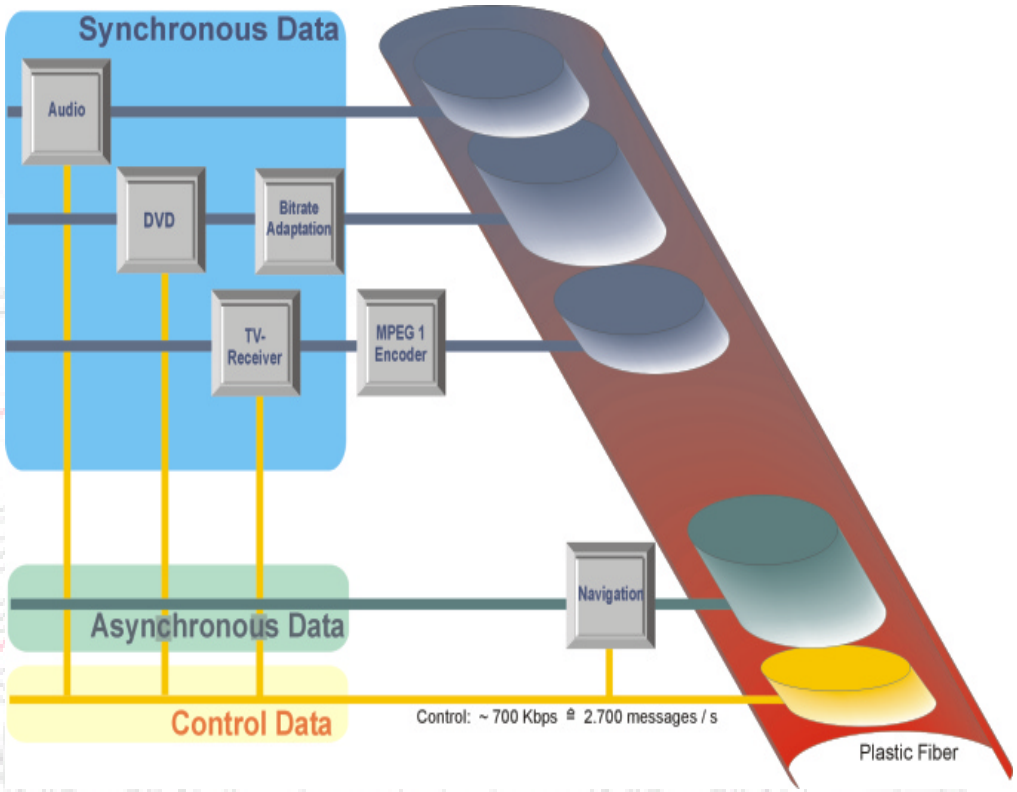
- **Communication paths become more complex**
- **Need to transmit control as well as data to all devices involved**
- **Costs of required cable connections grows exponentially**
- **Space / volume required for the cable connections grows exponentially**
- **Sharing limited number of user interfaces (screen, audio system, etc.)**
- **Data is transported between components in a closed system**

MOST: Media Oriented Systems Transport

-Optical fiber physical layer

-Complete framework for control of Distributed "In-Vehicle" Networks





Adoption in Automotive Industry

Agenda:

- Technology Background
- Adoption in Auto Industry
- Market Outlook
- Content Protection

MOST Cooperation

- Founded by BMW, DaimlerChrysler, Harman Becker and Oasis SiliconSystems
- Proliferates Technology across automotive industry
- Outreach to other industry organizations (AMI-C, CEA, OSGi, Bluetooth, etc.)

MOST is an open standard

- No license fee or royalties
- Non-assertion licensing agreement with MOST CO
- Same agreement signed by all members
- Released Specification available to the public
- MOST CO members drive the standard
- Widely accepted within the automotive industry

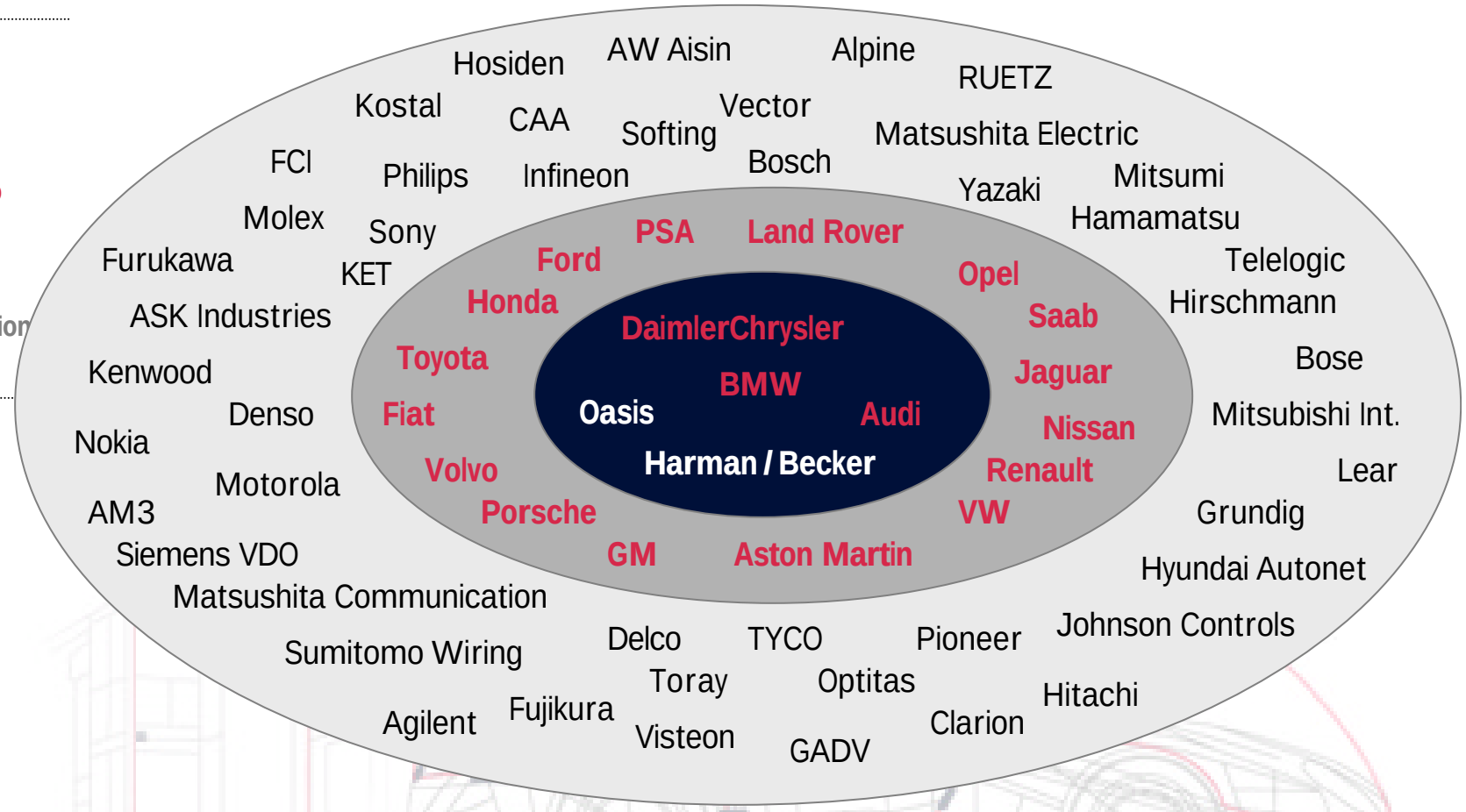
More information on MOST

- www.mostcooperation.com

Adoption in Automotive Industry

Agenda:

- Technology Background
- **Adoption in Auto Industry**
- Market Outlook
- Content Protection



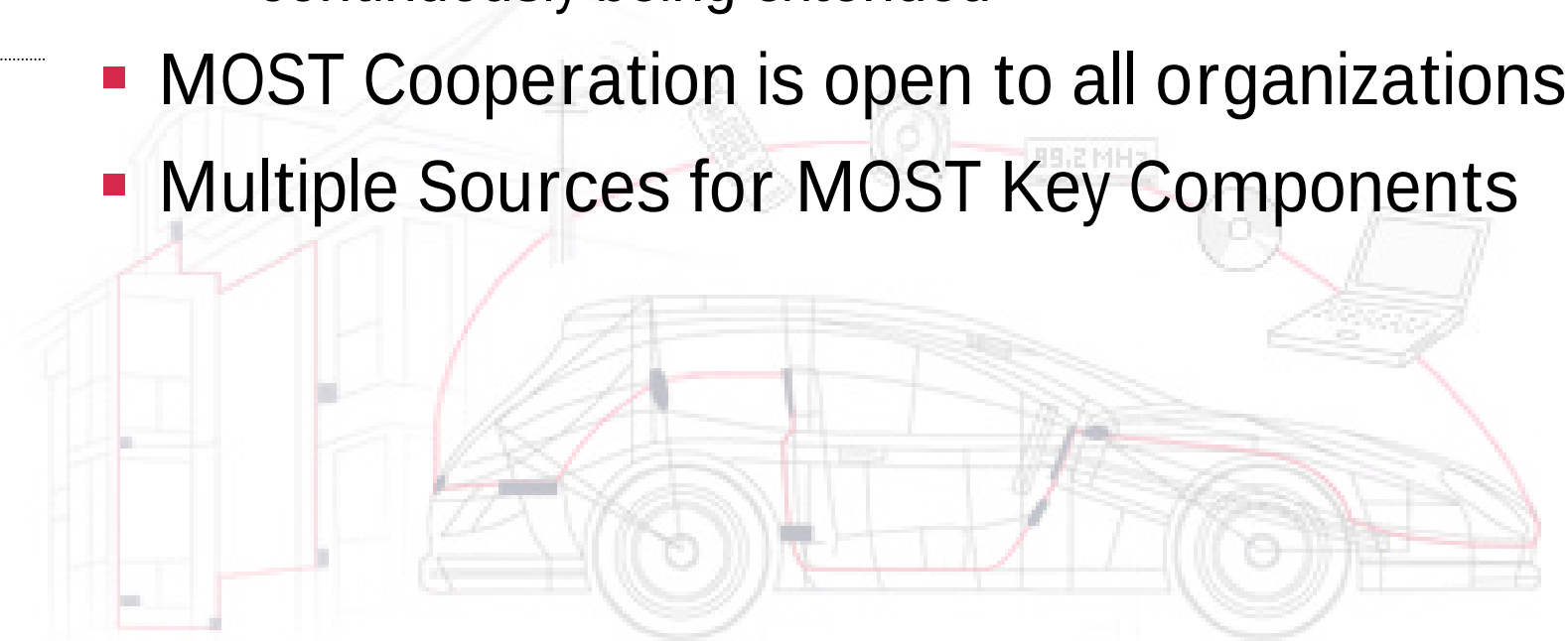
Status: October 20, 2002

Adoption in Automotive Industry

Agenda:

- Technology Background
- **Adoption in Auto Industry**
- Market Outlook
- Content Protection

- MOST Technology is in volume products
- The Specifications of the standard are
  - publicly available
  - Comprehensive & complete
  - continuously being extended
- MOST Cooperation is open to all organizations
- Multiple Sources for MOST Key Components



BMW 7 Series

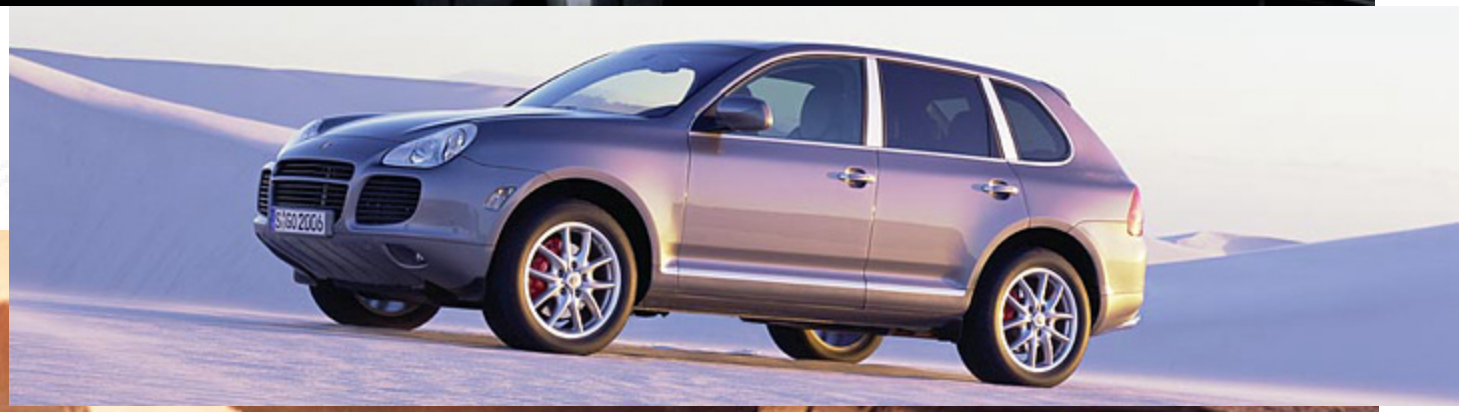
Mercedes E

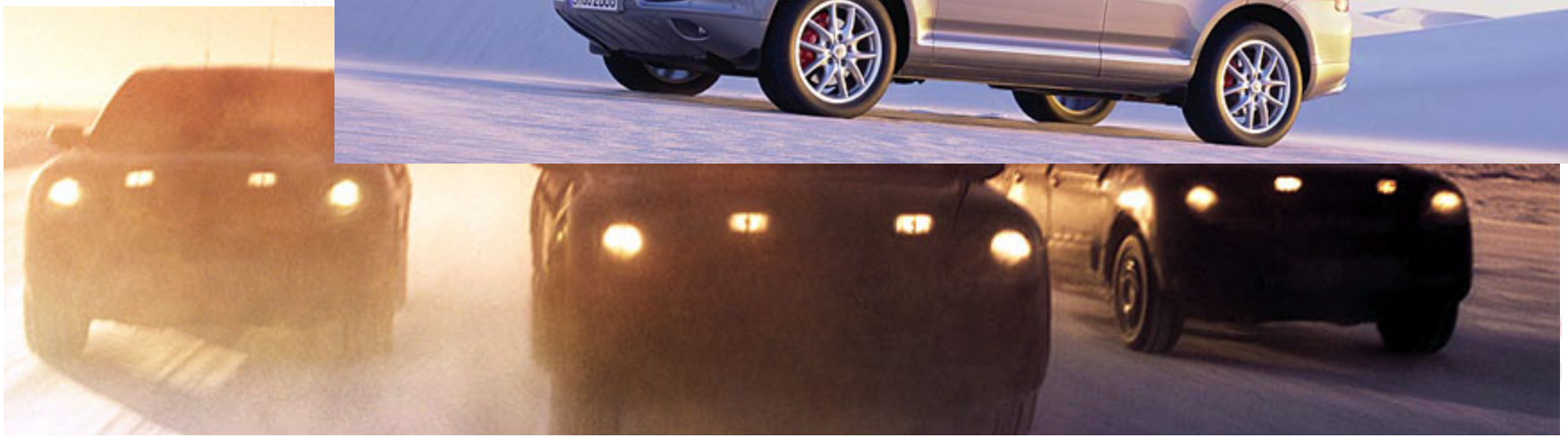
Lancia Phaedra

Saab 9-3

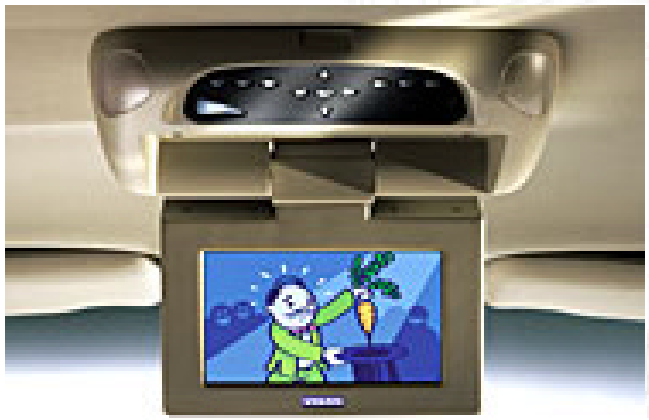
Audi A8

Porsche Cayenne

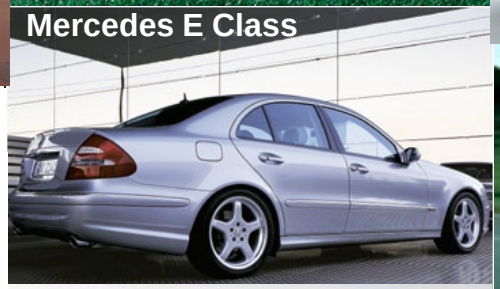




Volvo XC-90



MOST is on the road



Volvo XC90

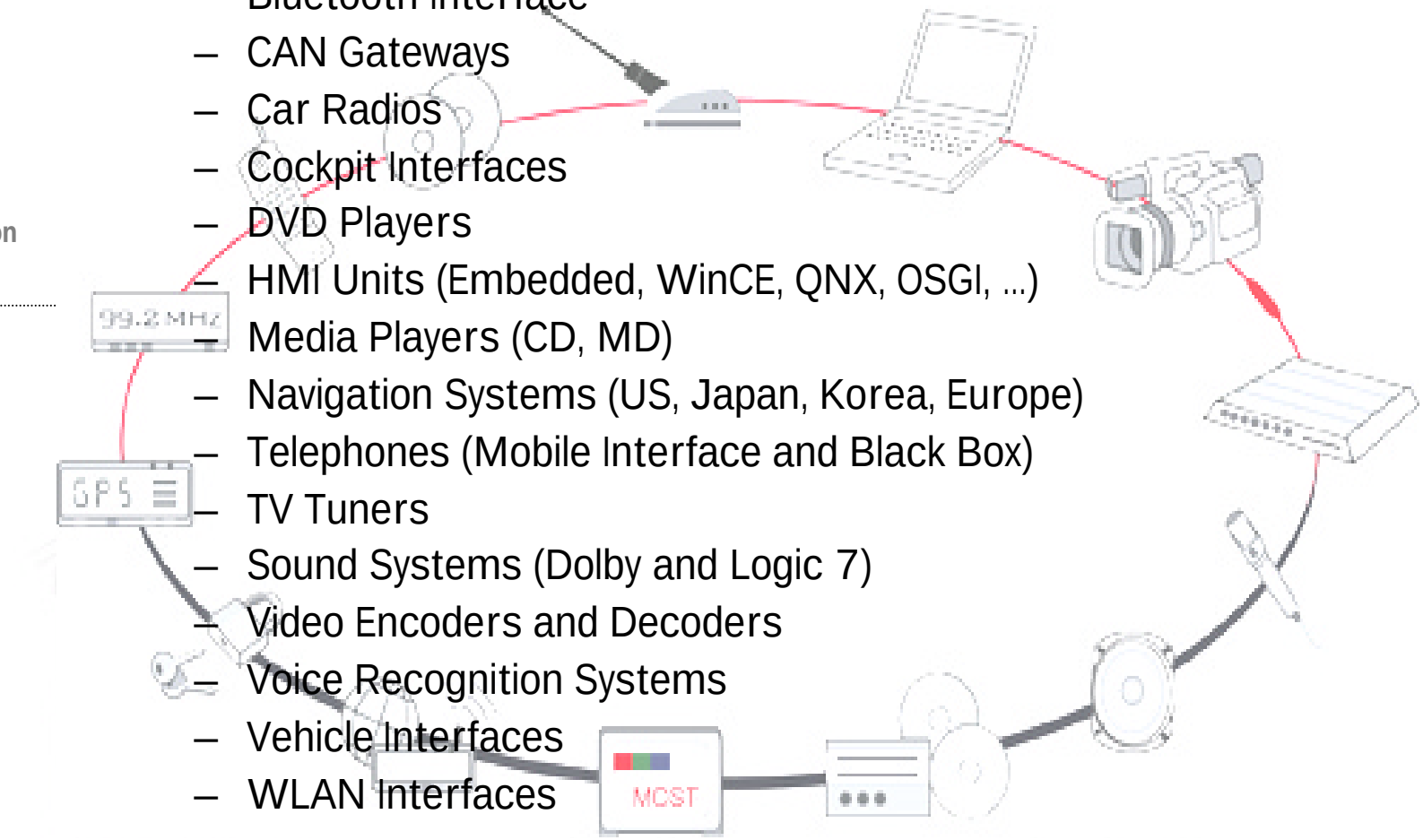
Saab 9^3

MOST technology is in many devices

Agenda:

- Technology Background
- **Adoption in Auto Industry**
- Market Outlook
- Content Protection

- More than 60 Devices with MOST IF in the Market
  - Amplifiers
  - Bluetooth Interface
  - CAN Gateways
  - Car Radios
  - Cockpit Interfaces
  - DVD Players
  - HMI Units (Embedded, WinCE, QNX, OSGI, ...)
  - Media Players (CD, MD)
  - Navigation Systems (US, Japan, Korea, Europe)
  - Telephones (Mobile Interface and Black Box)
  - TV Tuners
  - Sound Systems (Dolby and Logic 7)
  - Video Encoders and Decoders
  - Voice Recognition Systems
  - Vehicle Interfaces
  - WLAN Interfaces



Market Acceptance

Agenda:

- Technology Background
- Adoption in Auto Industry
- **Market Outlook**
- Content Protection

- More than 3500 development platforms shipped
- More than 3.5 Million MOST nodes produced
- Volume developing better than originally projected
- More than 10 Million nodes / year in 2004

Year	2000	2001	2002	2003	2004	2005
Thousands of Nodes	5	300	3500	5000	10000	15000

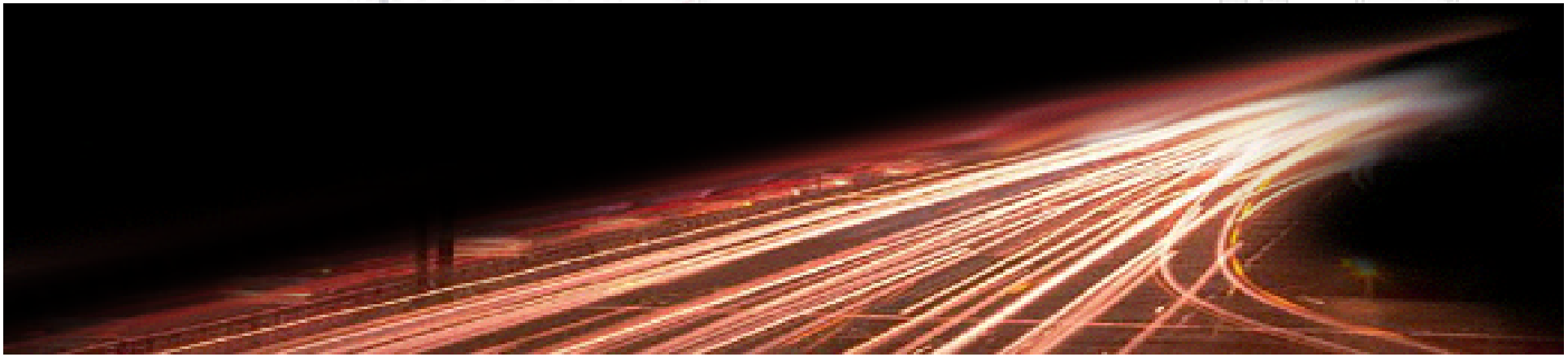
Automotive Mass Market

MOSTCO is...

- ... more than 70 member companies
- ... more than 10 vehicle models on the road
- ... more than 60 devices in volume production
- ... more than 3.5 Million nodes already produced
- ... more than 500 man years in development and verification of Specifications
- ... more than 10,000 engineers involved
- ... a huge worldwide commitment

Agenda:

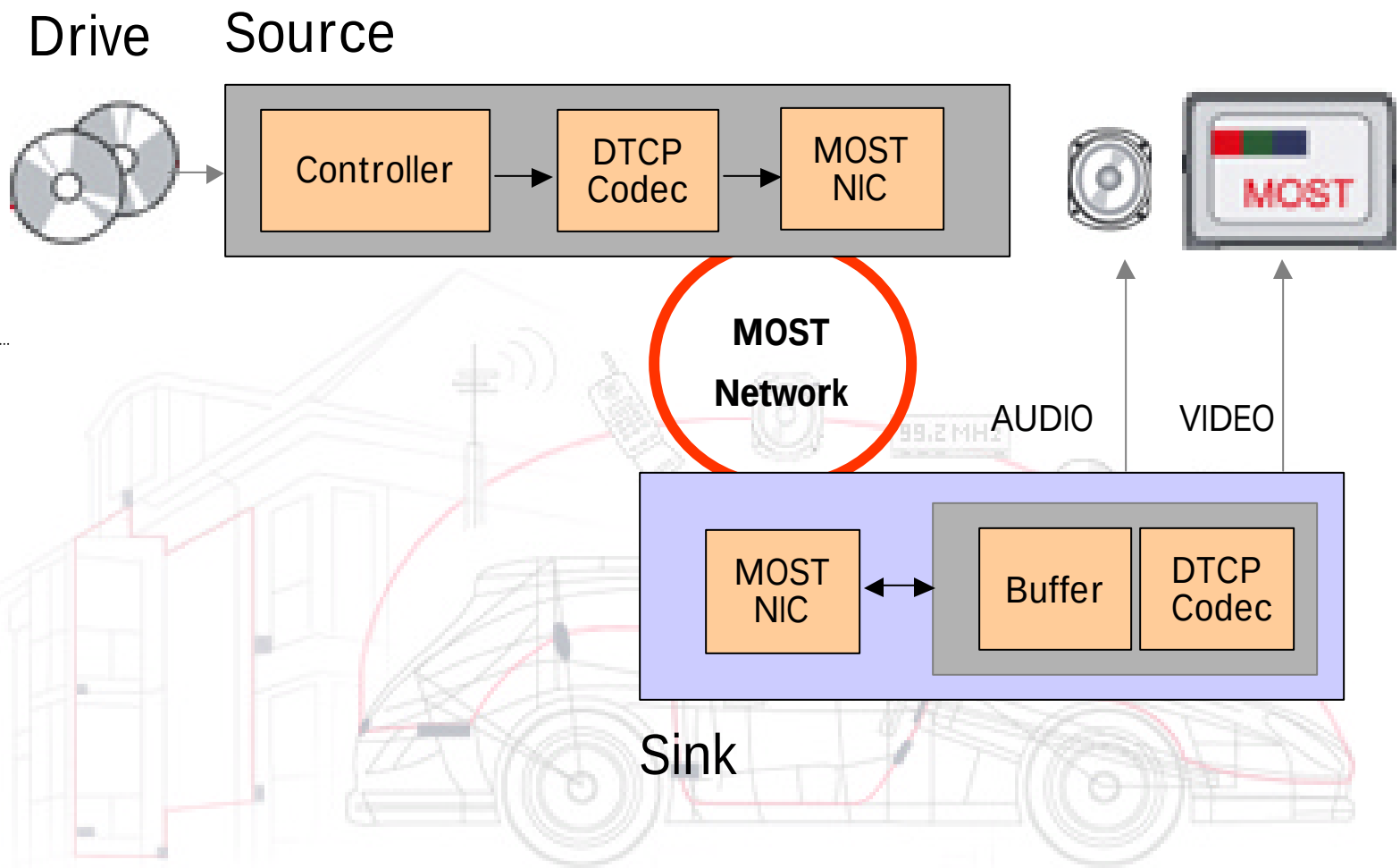
- Technology Background
- Adoption in Auto Industry
- **Market Outlook**
- Content Protection



DTLA approved DTCP over MOST

Agenda:

- Technology Background
- Adoption in Auto Industry
- Market Outlook
- **Content Protection**



Content Protection

Agenda:

- Technology Background
- Adoption in Auto Industry
- Market Outlook
- **Content Protection**

- DTLA has approved use of DTCP over MOST
- DVD-Audio uses DTCP digital outputs
- Proposal has been submitted to CCA to add MOST to list of digital video outputs
- Carmakers and automotive industry strongly support MOST for in-vehicle networking and the proposal submitted to CCA
- Industry is working to include accepted content protection methods
- MOST is the choice for distributing entertainment and information data within the vehicle
- MOST provides a platform for OEM and aftermarket infotainment systems

Annex 8

2000-01-C028

Future Automotive Multimedia Subsystem Interconnect Technologies

Visteon Automotive Systems

ABSTRACT

For the past decade or so, automotive entertainment subsystem architectures have consisted of a simple Human Machine Interface (HMI), AM-FM tuner, a tape deck, an amplifier and a set of speakers. Over time, as customer demand for more entertainment features increased, automotive entertainment integrators made room for new features by allowing for the vertical integration of analog audio and adding a digital control. The new digital control came to entertainment subsystems via a low speed multiplexing scheme embedded into the entertainment subsystem components, allowing remote control of these new features. New features were typically incorporated into the entertainment subsystem by independently packaging functional modules. Examples of these modules are cellular telephone, Compact Disc Jockey (CDJ), rear-seat entertainment, Satellite Digital Audio Radio System (S-DARS) receiver, voice and navigation with its associated display and hardware. Figure 1.0 is a block diagram of typical entertainment subsystem. This paper discusses alternatives to the module-expansion of entertainment subsystem via low speed digital control and analog audio. Moreover, the discussion is expanded to cover future multimedia and infotainment subsystem interconnects technologies.

INTRODUCTION

Recently, great achievements have been reached in information, communication, entertainment, comfort, safety and security products. Moreover, new Intelligent Transport Systems (ITS) services, requiring state-of-the-art electronics, are appearing on the market to help drivers process information, make decisions, and operate vehicles more safely and effectively.

As a consequence, our cars will be equipped more and more with digital systems communicating and exchanging information. Whenever possible, the trend is to go towards a super-integration of these systems, but a number of them will always be distributed in different locations of the car. A transport bus is necessary as a backbone for all the bit-streams and commands flowing between them. Table 1.0 lists some of the most important applications that are already present or will be soon introduced into the automotive environment.

Safety & Security	**Entertainment**	**Information and Communication**
Road-side assistance Mayday	Radio (AM/FM/DAB)	Internet access
Panic call	Audio (cassette, CD player, MP3)	E-mail
Collision avoidance	S-DARS	Weather forecast Head-line News Stock quotes Traffic updates Paging
Antitheft system	Video (TV, DVD)	Tourist information
Traffic information	Games	Navigation
Tolling system		Car diagnosis
		Mobile phone

Table 1.0: Near-Future Vehicle's Features

This paper assesses the suitability of current mobile multimedia transport for the accommodation of these technological advances. In addition, this paper identifies system and functional requirements for future mobile multimedia transport as well as differences between existing networks such as Ethernet, IEEE 1394, Media Oriented System Transport (MOST).

CURRENT MOBILE ARCHITECTURE

In the beginning of the automobile era, the primary function of a vehicle was a reliable transport. Over a period of years, the desire for a basic transport has been coupled with the desire for comfort, convenience, entertainment, information, communication, safety and security. Vehicle manufacturers made room for the new

features by allowing for the vertical integration of analog audio and adding a digital control. Despite the digital nature of most of the new added modules and the introduction of Digital Signal Processor (DSP) within the mobile multimedia system, the vertical integration of audio and its transport remained analog.

The Current mobile architecture with its analog transport has the following limitations:

- Analog transport complicates the subsystem interconnects, decreases reliability, adds weight and cost. Two twisted pairs are required for cabin media, one twisted pair is required for voice module, one twisted pair is required for cellular phone module, one twisted pair is required for navigation module, 3-5 wires are required for low speed multiplex scheme and synchronization. Additionally, two more twisted pairs are required for an optional media player such as CDJ. In the case of rear seat entertainment, more wires or coaxial cables are required for video. The number of wires or coaxial cables required for video applications is proportional to the number of rear seat occupants. Moreover, these wires require wide connectors with more pins at the analog power amplifier's input connector.

- The module level expansion strategy and vertical integration of analog audio resulted in a closed architecture with limited expansion path. The number of pins available at the power amplifier's input connector bounded the expansion path. In addition, the new subsystem had a short life and is not compatible with the digital trending of future entertainment features such as digital audio, digital video, Digital Audio Broadcast (DAB) and next generation of compact disc technology, Digital Versatile Disc (DVD).

- The module level expansion strategy and vertical integration of analog audio resulted in costly subsystem architecture. Often, modules added to the subsystem exhibited wasteful redundant hardware resources in order to achieve compatibility with an analog architecture. An example of this hardware wastefulness is the addition of a digital-to-analog converter to the output of CDJ or CD player to achieve compatibility with analog integration and processing. Moreover, the hardware resources available for each module are for that modules' own use and can't be shared with other subsystem's modules. This will add cost to each module and will contribute to the overall cost of the subsystem.

Presently, the module level expansion strategy and vertical integration of analog audio has reached its upper integration limit for an HMI, AM-FM tuner, voice, cellular phone, CDJ, a media player, Steering Wheel Control (SWC), a Rear Integrated Control Panel (RICP) and navigation. The implementation of such a subsystem requires seven modules and a minimum of 31 wires. A saving of one module and four wires is possible if a media, HMI, tuner, and power amplifier is packaged in one module. However, a complicated heat dissipation

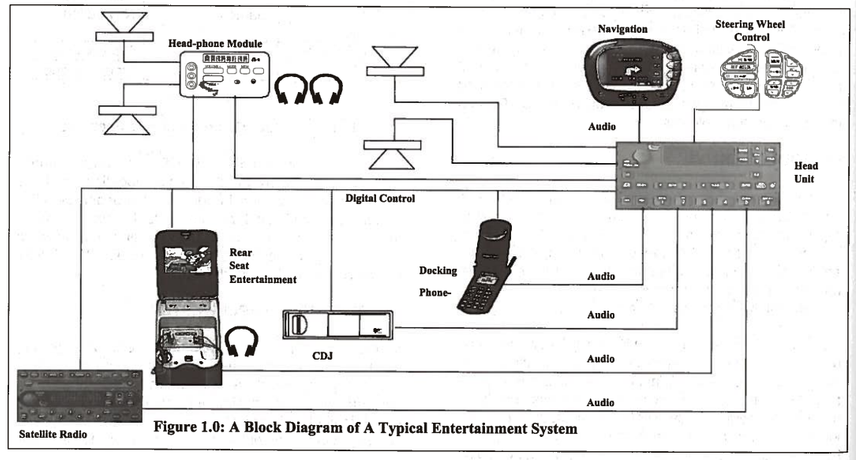


Figure 1.0: A Block Diagram of A Typical Entertainment System

and overall power management strategy is required for the success of such integration, and may limit the audio performance.

REQUIREMENTS FOR NEW TRANSPORT

Vehicles are running out of the real estate required to house new modules. Interconnect harness thickness and costs are ever increasing. This section is a discussion of both system and functional requirements for a new transport:

- Open Standard: the new transport shall be an open standard to ensure that all vehicle and electronic product manufacturers have equal access to the standard and to the market.

- Minimal Standard: the new transport standard shall be extensible and capable of migration to future technologies and different physical media with no impact on application software. A layered approach to the protocol, such as is used in the reference model of Open System Interconnect (OSI) model, shall be used.

- The new transport shall be digital to allow multiplexing of data, control, audio and video over the same media. In addition, digital transport enables open system architecture; a node can be added at anytime during the vehicle's life without modifying an existing node's connector. Moreover, a digital transport enables the natural transport of digital data from one ever increasing digital application to another without exhibited wasteful redundant hardware resources such as Digital-to Analog (DAC) and Analog-to-Digital (ADC) converters.

- Safety & Security: the new transport shall provide an environment into which devices can be plugged, unplugged, and operated in a vehicle in a manner which does not threaten the integrity of the vehicle or the devices that are being used. The new transport shall include suitable security for transactions involving the exchange of money and/or proprietary information.

- Manufacturability: the new transport compatible devices and software shall be easy to design, implement and integrate. This implies minimal specifications and unambiguous requirements. Ease of manufacture helps to assure the wide adoption of the standard and contributes to lowering the cost of the manufactured items.

- Ease of Use: the addition or removal of devices shall be easy enough for a consumer to accomplish the task with little or no expert assistance required.

- Low Cost: the incremental direct material cost to implement a new transport node shall be a fraction of the cost of the application supported.

- Graceful Degradation: the new transport's physical layer and its supported bus topology shall be designed such that any fault shall not cause any damage to the cable, the vehicle or any attached devices. Functional operation under any of these conditions may cease but shall resume within the boot/discovery time after the fault is removed. The new design shall be such that no single failure, other than a fault in the physical layer as described above, or the loss of primary power, shall cause the entire system to fail.

- Hot Plug and Play: it shall be possible to attach devices to, and remove devices from, the new transport system at any time, whether power is on or off.

- Self-Identification: devices attached to the new transport shall be able to identify themselves to other system devices and shall self configure to obtain unique addresses on the new transport. No user intervention shall be required to complete this configuration other than the provision of the appropriate application software.

- Short Boot/Discovery Time: the new transport and all attached nodes shall complete self-configuration within one second after device initialization. When a new node is added to the system while it is operating, detection of the new node and reconfiguration of the system to include it shall be completed within 2 seconds.

- Peer-to-Peer Communications: the set of devices that is likely to be attached to the new transport is unpredictable and no single device is guaranteed always to be present. Therefore, a device on the new transport shall be able to communicate directly with any other device in the system and the vehicle without need for any additional device. An application that is implemented across multiple devices may choose to implement a "master controller" for that application but this shall not be a requirement for all applications.

- Automotive Physical and Electrical Specifications: the new transport's physical layer shall meet automotive environmental (temperature, vibration, shock, EMI, etc.) and electrical specifications (reverse voltage, load dump, etc.) for the environments in which the devices are installed, e.g., passenger compartment, trunk, etc.

- Extensibility: The new transport shall be extensible to

accommodate evolving technologies and new applications. A layered approach to the protocol is required to guarantee minimum impact on existing designs as new physical layers and new applications are developed.

- Security and Authentication Services: The new transport protocol shall provide security and authentication services for access to vehicle functions. It is anticipated that additional services will require additional security measures to accommodate applications or transactions that require billing, authentication, confidentiality, confirmation, non-repudiation, etc.

- Bit Error Rate: the bit error rate shall be less than 1 x 10^{-9}. Applications requiring better than this shall be able to implement appropriate measures at higher layers of the protocol.

- Time-Critical Delivery of Packets- Deterministic Latency: Some applications may require that messages be transmitted within a given time period. The new transport shall be deterministic and it shall be possible to determine the maximum latency for any message in a given system configuration

- Private Message Service: Equipment manufacturers wish to be able to develop applications that span their own suite of products and provide competitive functions and features not achievable when products from different manufacturers are interconnected. The new transport protocol shall support the implementation of private messages that will allow such applications to be developed.

- Power Loading: the new transport gateway shall make power available for all devices connected to the new transport system. The total operating current drain of all devices connected to the new transport shall not exceed the capacity of the gateway unless power is routed directly to the device from another source or the device is self-powered (e.g., internal batteries).

- Internetworking: it is anticipated that wireless access to and from the Internet will be required for many devices attached to the new transport system. The new protocol shall not preclude the future implementation of internetworking services across multiple gateways or bridges between the new transport and other subnets such as Intelligent Transportation Data Bus (IDB).

- Explicit Device Addressability: it shall be possible to send a message from a device to a specific other device. It shall be possible for the sender to determine and use the desired recipient's unique address to deliver this message and all other devices shall ignore it.

- Broadcast Messages: it shall be possible for an application to generate a broadcast message to all devices connected to the new transport without having to address each one explicitly. A broadcast message may or may not require acknowledgment or confirmation of delivery. For example, an application may require confirmation (acknowledgment) that at least one receiving device capable of acting on the message has received the message.

- Consumer-Friendly Device Connection (No Special Tools Required): in most cases, it shall be possible for a consumer to install new transport nodes with common hand tools. There will be cases where professional installation may be required, but this should be the exception, not the rule.

- Wake/Sleep: any node shall have the ability to wake up the new transport system or put it to sleep. It shall be possible to wake up the nodes by sending a wake-up message, such as a pager. It shall be possible to put any node and all connected devices back to sleep with a sleep message. Absence of message traffic on the new transport for more than 30 minutes shall cause the new transport and all attached devices to go to sleep.

- Priority Sensitive Flow (Isochronous): consumer electronics devices such as video games, DVD and MP3 players, Dolby AC-3 audio components, etc., may require support for high speed isochronous data communications (i.e., data packets delivered at a guaranteed rate in a guaranteed order).

- Data Types: The new transport protocol shall allow any data type (ASCII, binary, bulk, etc.) to be transmitted in a message without need for any special escape characters or other similar artifacts added by the application.

- Fair Access to the new transport system: no single device shall be allowed to monopolize the new transport system.

- Message Priority Flagging: the new transport protocol shall provide a means to specify that the current message is a high or normal priority message. The protocol simply provides a mechanism to identify the message as a high priority message. It is up to the device manufacturer to determine whether support is provided and up to the application software to determine what action is to be taken when a high priority message is received or presented for transmission.

- Confirmation of Message Delivery, if required: a device sending a message to another device shall be able to explicitly request confirmation of error-free delivery of that message from the receiving device.

EXISTSING PROTOCOLS: ETHERNET, IEEE 1394 AND MOST

Fast Ethernet

Ethernet is a term commonly used to describe a variety of network implementations that share the same basic technology. Some early varieties of Ethernet are 10Base-2 and 10Base-5 which are also called 'thin net' and 'thick net' respectively. All nodes on such network tap into a single cable. A later version of Ethernet, 10Base-T, introduces the concept of a hub or a repeater. All nodes are connected directly to a single repeater, which simplifies cabling and provides buffering of electrical signals.

A newer version of Ethernet, called Fast Ethernet, operates at 10 times the speed of 10Base-T or 100 Mbit per second. It has the same star topology as 10Base-T and comes in a few different versions called 100Base-TX, 100Base-FX, and 100Base-T4. The difference between these versions is the physical layer which is electrical for TX and T4, and optical for FX. In this paper, Fast Ethernet will refer to the most common version 100Base-TX.

Ethernet Topology

Thick net Ethernet uses a single cable as a backbone for the network. Each node in the network taps into this cable through what is called a T connector. In an office environment, this cable could be routed through the ceiling with taps dropping into each office. This works well except it is difficult to add new users to the network and signal quality is sometimes difficult to control.

A Fast Ethernet uses a central repeater which connects directly to each node. This star topology enables the signal quality on the transmission line between a computer and the repeater to be well controlled and provides a relatively simple means to add users. The repeater has a number of ports which connect to each computer on the network. To add another computer, a wire is run from a free port on the repeater to the new computer. If there are no free ports, typically, another hub (or switch) can be connected to an uplink port to expand the network to virtually any size.

Ethernet Physical Layer

In a thick Ethernet, all computers are connected to one coaxial cable. This cable is used for sending and receiving messages. When the bus is idle, the voltage on the coax cable remains in a high impedance state at an intermediate level. This level is not a one or a zero, so that all nodes can easily determine if the bus is idle. When a node is transmitting, the voltage on the bus is pulled to high and low voltages depending on the data to be transmitted. Only one computer can send information at any one time. If multiple nodes try to transmit at the same time, a collision occurs, the data from both computers is corrupted, and both computers have to stop transmitting and try again when the bus is idle.

In Fast Ethernet, all nodes connect to a central repeater through two sets of twisted pairs. One pair is for transmitting and the other for receiving. Although each node has its own cable, the network operates exactly like thick Ethernet at a higher level. When a computer sends a message to the central repeater, the repeater sends the data exactly as received to all other computers connected to it. Again, if two computers try to send at the same time, a collision occurs, and both computers must try again later.

Ethernet Arbitration

Since all the computers on an Ethernet share the transmission media, only one computer can send information at any one time. If multiple nodes try to transmit at the same time, they must arbitrate for use of the bus. The rules that every node follows is called Carrier Sense Multiple Access with Collision Detection (CSMA/CD). Before a node sends a message on the bus, it sends a stream of one's and zero's called a 'carrier'. All other nodes on the network sense this carrier and do not attempt to send their own message until the original node completes its message.

There is a finite amount of time from when a node begins sending a carrier to when all nodes detect this carrier. During this time, other nodes may attempt to send a carrier. If this happens, a 'collision' will occur which corrupts the data on the bus. Transmitting nodes will both 'detect' this condition and stop transmitting. The rules then specify that each node must wait a random amount of time before attempting to transmit again. The probability that another collision will occur is low.

Ethernet Switches

Ethernet switches have become more popular than repeaters in recent years. An Ethernet switch is a more sophisticated device which prevents all messages from being sent to all computers connected to the switch. When a Fast Ethernet is implemented with a repeater, all computers connected to the repeater are said to be on the same collision domain. This is because a repeater

operates just like the coaxial cable in thick Ethernet. If two computers try to send messages at the same time, the messages collide.

An Ethernet switch divides the network into many collision domains. Each port on a switch is a different collision domain. For example, if one computer is connected to a port on an Ethernet switch, the collision domain consists of two nodes; the computer and the switch. If a port on a repeater is connected to a port on a switch, the collision domain consists of the switch and all computers connected to the repeater.

The switch has intelligence which learns the addresses of the computers connected to each port. When a message is received at one port, the destination address is determined and the message is sent out the appropriate port. Switches are typically much more efficient than repeaters. However, they cost more.

Ethernet Communication Mechanism

Information in Ethernet networks is communicated in packets. Each packet consists of a header, usable data and a checksum. The header contains information such as source and destination address, the length of usable data and possibly information about the message type. The checksum is a code sent at the end of the message so that the receiving node can determine if the packet was corrupted during transmission.

Since the header and checksum are only used to send the packet safely from the transmitting node to the receiving node, it is considered network overhead. It is not information usable by the application. In Fast Ethernet, this consumes 18 bytes. If you include the arbitration time, the total overhead is 38 bytes. In addition, the minimum usable data is 46 bytes per packet. Even if you only wish to send one byte, you still must send the 18-byte header, 46 bytes of data, and wait 20 bytes worth of time for the bus.

The efficiency of the network can be defined as the number of user data bytes per packet divided by the number of bytes in the packet plus the overhead of waiting for the bus. If only one byte of user data is sent per packet the efficiency is 1/(64+20) X 100% = 1.2%. Since the maximum user data per packet is 1500 bytes in 100BaseT, the theoretic maximum efficiency is 1500/(1518+20) X 100% = 97.5%.

The maximum efficiency is never achieved since many collisions will occur if there is a lot of activity on the bus. The effect on efficiency is difficult to predict.

Ethernet Audio Example

Let's consider a PC with a sound card connected over the network to a server which stores WAV files. A particular Wav file from the server can be played on the sound card in the following way. The client software on the PC manages a FIFO (First-In First-Out) which continually outputs audio data to the sound card. When the FIFO gets close to being empty, the client sends a message to the server to send more data. The server sends another packet to the client to fill the FIFO up again. As long as the server sends the new packet before the FIFO empties, audio can be heard. If the server responds slowly or the network is very busy, the packet may not arrive in time, the FIFO will empty and sounds will momentarily stop. This is unacceptable for most audio applications.

There is a trade off between FIFO size, the frequency of requests for more data and packet size. Since the large packets are more efficient than small packets (% overhead from header, etc), let's assume we will use the largest packet size; 1500 bytes of user data. If the audio sample rate is 48 kHz, and the audio is 16 bits/sample stereo, then we need an average of 192K bytes/second or 128 packets/sec. The overhead for the header, checksum and the required idle between packets is 38 bytes. Since the client software on the PC with the sound card must inform the server when the FIFO is nearly empty, there are another 84 bytes of overhead to send this message to the server.

The minimum total bandwidth required for one audio channel is:

1500 + 38 + 84 = 1622 bytes/packet

1622 bytes/packet * 128 packets/sec = 207616 bytes/sec

207616 bytes/sec * 8 Bits/bytes = 1.66 Mbit/sec

Since the packet size in this example is the largest allowed by Fast Ethernet, the network overhead is small compared to the audio data throughput. The disadvantage of the large packet size is the buffer size requirement in the Client. It must be 1500 bytes deep plus more for handshaking. If the extra depth is not large enough for the network to guarantee another packet will arrive prior to the buffer emptying, a loss of audio quality may occur. If the buffer empties, the audio stops. In a Fast Ethernet, it is impossible to guarantee any bandwidth. If the network has lots of traffic, you may not even be able to get the 1.66 Mbit/sec average throughput that is required. More commonly, at times of high traffic the buffer may empty no matter how large it may be.

IEEE 1394

The IEEE 1394 specification is a hardware specification that defines the serial bus architecture that Apple computer initially named FireWire. It defines serial bus specific extensions to the Control and Status Register (CSR) Architecture for Microcomputer Buses formally adopted as ISO/IEC 13213 (ANSI/IEEE 1212).

This architecture defines a set of core features such as node architecture, address space, common transaction types, Control and Status Registers (CSR), configuration ROM format and content, message broadcast mechanism to all nodes and interrupt broadcast to all nodes. IEEE 1394 specifies how units attached to a serial bus can talk to each other, but does not define the protocols used to communicate between the nodes.

IEEE 1394 is similar to Fast Ethernet in many ways. Data is always communicated between nodes in packets. If multiple nodes try to send packets at the same time, they must arbitrate for the bus. The information in the packet headers, the packet sizes and the arbitration method, are different. However, the fundamental mechanisms are similar.

The most significant feature that IEEE 1394 provides (which Fast Ethernet does not) is guaranteed bandwidth for real time applications. These applications are allocated isochronous bandwidth which enable real time data to be communicated in packets sent at regular time intervals. This is an improvement over Fast Ethernet. However, it will be shown that there are still serious limitations.

The raw bit rate for IEEE 1394 is defined to be selectable between approximately 100, 200, and 400 Mbit/sec. Work is currently being done on an 800 Mbit specification as well. Silicon is currently being advertised to run at both 100 and 200 Mbit. However, most implementations are now at 100 Mbit/sec which is the same data rate as the large installed base of 100BaseT. Gigabit Ethernet is currently under development as well.

IEEE 1394 Physical Layer

The physical layer for IEEE 1394 consists of two sets of twisted pair wire for signals and two wires for power and ground connected between each pair of nodes. One set of twisted pairs is called data and the other is called strobe. When one node begins to send a packet, it sends Nonreturn-to-Zero (NRZ) data on the data line and transitions the strobe line only between consecutive 1's or 0's. Both sets of twisted pairs are bi-directional. Each node sends and receives data on the same sets of wires. When neither node is sending data, the twisted pairs are held in a high impedance state.

In contrast, the physical layer for Fast Ethernet consists of two sets of twisted pairs; one pair is used for transmitting data and the other pair for receiving data. This approach makes recovering data from the transmission line slightly more difficult than with IEEE 1394 since there is no strobe signal. An advantage of keeping the transmission lines unidirectional, however, is that the transmission line can be longer. The maximum cable length for Fast Ethernet is 100meters, while it is only 4.5 meters for IEEE 1394.

Each node on an IEEE 1394 bus (or in a Fast Ethernet) has its own timing source which is typically a crystal oscillator. This timing source is used by a node to transmit data and is used by a node to over sample the received data and strobe lines to recover the data. This means that all IEEE 1394 nodes are asynchronous at the lowest level. The accuracy of the timing reference in IEEE 1394 is specified to be ± 100 PPM which is typically the frequency tolerance of widely available crystal oscillators.

The nominal data rate in 100 Mbit IEEE 1394 is 98.304 Mbit. If the crystal oscillator at a particular node is operating at the high end of its frequency tolerance, it will be able to transmit data at 98.304 Mbit +100 PPM or 98.314 Mbit/sec. If it is operating at the low end of the frequency tolerance, it will transmit data at 98.294 Mbit/sec. This may seem like a trivial issue. However, the section on system timing will illustrate some important consequences for real time applications.

IEEE 1394 Topology

The physical topology for a typical IEEE 1394 network is a tree structure. Typically, a node will have a least two ports which enables multiple nodes to be daisy chained together. If a node has more than two ports, multiple branches can be created. During initialization, one node is defined to be the root node with all nodes extending down different branches. The topology can have any number of branches if no loops are created.

The tree topology, with the ability to daisy chain nodes, has the advantage of simplicity for small networks. If you have a few devices, it is easy to plug them together in a daisy chain. However, large networks can be cumbersome, particularly if network performance is optimized. To improve performance, it is desirable to minimize the propagation delay of data between any two nodes in the network. Long daisy chains can be split into many branches to reduce the delay; however, care should be taken to balance the length of the branches.

In a Fast Ethernet, a repeater or switch is required for a network with more than two nodes. This makes small networks complicated. However, it makes large networks simpler.

IEEE 1394 Arbitration

As described previously, all computers on a Fast Ethernet using a repeater (not a switch) have the same collision domain. If multiple nodes attempt to transmit at

the same time, the transmitted data is corrupted, this condition is detected, and the nodes begin to arbitrate for the bus. Likewise, all nodes on an IEEE 1394 network have the same collision domain. Only one node can send a message at one time. If multiple nodes try to send messages at the same time, only one node will gain control of the bus.

In a Fast Ethernet, if a collision between two nodes occurs, both nodes will stop transmitting and wait a variable amount of time before another attempt. If they both happen to wait the same amount of time, another collision will occur. Mechanisms are built into the network to minimize the probability of nodes colliding more than once or twice.

An IEEE 1394 network operates differently. Nodes which are closer to the root node have a higher natural priority. When two nodes attempt to transmit at the same time, the node with the higher priority wins arbitration and control of the bus. In order to prevent higher priority nodes from monopolizing the bus a fairness interval is defined. During a fairness interval, all nodes are given the opportunity to send one message.

Unlike Fast Ethernet the arbitration mechanism in IEEE 1394 is deterministic. There is zero probability that nodes will collide many times before successfully sending a message. This is important for the delivery of real time data since any unpredicted delay, no matter how unlikely, may cause buffers to overflow or underflow.
The arbitration process in IEEE 1394 consists of bus request/grant handshaking between child and parent nodes. A parent node is defined as the node on a 1394 cable which is closer to the root. The node which is further from the root, is called the child. If a child and a parent both request the bus at the same time, the parent will block the child's request and send its request to its parent. The request continues down the tree until it reaches the root node. The root node issues a bus grant which travels back up the tree to the node requesting control of the bus. Once a node receives a bus grant, it can begin sending a message.

The time that it takes to arbitrate for the bus depends on the size of the network. A bus request from a node at the end of a number of branches must propagate down the tree to the root and back up the tree to the requesting node. Information must be sent down all other branches to prevent any other node from driving the bus (through a bus request), until the granted message is sent. The total arbitration delay for a network with N hops (from parent to child or child to parent) between the furthest two nodes, is about N x 80ns in a 100 Mbit IEEE 1394 network.

IEEE 1394 Communication Mechanisms

While Ethernet treats all packet data the same, IEEE 1394 provides different types of packets. The primary packet types are asynchronous and isochronous. Asynchronous packets are functionally equivalent to Ethernet packets. Isochronous packets are only available in IEEE 1394, and provide guaranteed bandwidth to time critical applications.

IEEE 1394 Asynchronous Packets

An asynchronous packet consists of a header, a header checksum, user data and a user data checksum. The header contains information such as source address, destination address, message length, message type, etc. The size of the header and the checksums is typically 24 bytes long. The user data can be up to 512 bytes long in a 1394 network operating at 100 Mbit/sec.

When a particular node receives a message, an acknowledgement signal is automatically sent back to the sending node. If the sending node does not receive an acknowledgement signal within a specified time, it will try to re-send it a specified number of times.

The total time required for sending a minimum size message consists of the arbitration time, the time to send the packet, the time to wait for an acknowledgement, the time for the acknowledgement message and a sub-action gap time. The sub-action gap time is the time required by a node to wait after the bus is idle before it can issue a bus request. The sub-action gap time is required to ensure that no node will issue a bus request before the previous acknowledgement is received.

The total time for a one byte message in a large network (16 hops) running at 100 Mbit is about 1.4 µsec for arbitration, 2.2 µsec to send the message (28 bytes * 8 bits/byte * 10ns), and 5.6 µsec for acknowledge and sub-action gap. The total time is about 9.2 µsec. This means the efficiency is about 80 ns / 9.2 µsec X 100% or 0.9%. The total time to send a maximum size message (512 bytes) is about 50 µ sec with an efficiency of about 85%.

IEEE 1394 Isochronous Packets

The most significant improvement of IEEE 1394 over Ethernet for real time audio, video and voice applications is 1394's isochronous channels which provide guaranteed bandwidth to applications. When an application is allocated an isochronous channel, it is guaranteed to be able to send isochronous packets at regular intervals to any other nodes in the network. Each node in the network has a counter which defines these regular intervals. The Isochronous Resource Manager sends a message periodically to synchronize the counters in all nodes.

The Isochronous Resource Manager has a local counter which is clocked by a local 24.576 MHz clock and counts up to 3072. When the counter reaches 3072, the Isochronous Resource Manager resets its counter and broadcasts a message to all the nodes in the network to reset their local counters. The frequency of this synchronization message is 24.576 MHz / 3072 or 8 kHz.

After the synchronization message, all nodes that have been allocated isochronous bandwidth are allowed to send their packets. After all isochronous packets have been sent, then nodes that need to send normal asynchronous packets are allowed to do so. The time between the 8 kHz sync messages is 125 µsec. The maximum time that can be allocated to isochronous data is 100 µsec. This ensures that some bandwidth will always be available for some asynchronous packets.

IEEE 1394 Audio Example

The concept of an isochronous channel resolves the problem illustrated in the previous example of communicating audio over Fast Ethernet. Since a node that is allocated isochronous bandwidth is guaranteed to be able to send a packet every 125 µsec, the FIFO described in the Fast Ethernet example will not empty due to congestion on the network. In addition, since the audio data packets are guaranteed to arrive at fixed rate, it is intended that handshaking messages are not necessary. These are required in the Fast Ethernet example since the server needs to know when the FIFO in the PC with the sound card is nearly empty.

To send 16 bit stereo audio data at a 48 kHz-sample rate over an isochronous channel requires 24 audio data bytes per isochronous packet. This is shown below:

48 kHz * 2 bytes/sample * 2 channels = 192K bytes/sec

192k bytes/sec / 8k packets/sec = 24 bytes/packet

The header for an isochronous packet consists of eight bytes which specify the length of the data (in our case 24 bytes), the channel number, etc, It also provides a checksum for just the header. The checksum for the data consumes another 4 bytes. In total, there are 12 bytes of header and checksum overhead to send the 24 audio bytes.

Additional overhead comes in the form of arbitration time, gap time between packets and packet start and end symbols. The gap time and start and end symbols consume a fixed time equivalent to the time it takes to transmit about 3 bytes in 100 Mbit IEEE 1394. The arbitration time depends on the size of the network. For a large network that has 16 nodes between the furthest nodes, the arbitration time is approximately equivalent to 20 bytes of data.

In summary, to send the 24 bytes of audio data, a 1394 network consumes up to 35 bytes of overhead. The total time required sending such as message is about 4.7 µsec and the efficiency is about 40%. Since the maximum amount of time that can be allocated to isochronous is 100 µsec/Frame, the maximum number of audio channels that a 100 Mbit IEEE 1394 network can support is about 20.

Voice is typically sampled at 8 kHz with 8-bit resolution. The minimum data field in an isochronous packet is 4 bytes; so the packet size will be at least 16 bytes. Including the overhead described above, the time required to send this message is 4.1 µsec and the efficiency is about 2%. The total number of voice channels that a 100 Mbit IEEE 1394 network can support is about 24.

If 100 µsec per frame are consumed by isochronous packets, 25 µsec will be left for asynchronous messages. As shown in the previous section, the time to send a minimum size packet is 9.2 µsec. Consequently, a maximum of two asynchronous packets can be sent per isochronous frame. In other words, 16000 relatively small asynchronous packets can be sent per second.

As shown earlier, a 512-byte packet (largest possible in a 100 Mbit IEEE 1394 network) takes 50 µsec to send. Since only 25 µsec are available per frame for asynchronous messages, it takes two frames to send such a message. The message rate will be 4000 Bytes/second. It is possible to send such message, since the isochronous packets on the second frame are delayed until after the large asynchronous packet has been sent. If the transmit time of a packet were allowed to be larger than 50 µsec or if less than 25 µsec per frame were reserved for asynchronous packets, delivery of all the isochronous packets could not be guaranteed.

IEEE 1394 System Timing

The timing source for each node in an IEEE 1394 network is typically a crystal oscillator. As described in the physical layer section, the frequency of crystal oscillators can vary by ± 100 PPM from their nominal value. At every node a local 24.576 MHz clock is generated to clock the modulo 3072 counters used to create the 8 kHz isochronous frames. The Isochronous Resource Manager must send a synchronization message every 125 µsec to resynchronize the counters in all nodes since they are all clocked by slightly different frequencies.
If the Isochronous Resource Manager has a crystal operating slightly faster than another node, the modulo 3072 counter in the other node will sometimes count to

3072 between synchronization packets and sometimes will count to 3071. The most significant bit of this counter can be used as a synchronization signal. Its frequency will be the same as that of the Isochronous Resource Manager. However, the period will sometimes be 3071 x (1/24.576 MHz) and will sometimes be 3072 x (1/24.576 MHz).

If the crystal frequency of the Isochronous Resource Manager is slightly slower than another node, the counter in that node will sometimes count to 3072 and sometimes to 3073 (rolls over to 1) between synchronization packets. Again, the frequency of the MSB of that counter will be the same as that of the Isochronous Resource Manager (8 kHz). However, it will have 40 ns of jitter. The frequency of the jitter varies depending on the frequency difference between the two crystals. Depending on the frequency, which is unpredictable, this amplitude jitter can be clearly audible in high fidelity applications.

IEEE 1394 Computer Audio Example

Let's go back to our example of a server with a WAV file and a client PC with a sound card. If these two computers are connected together with IEEE 1394 network, then isochronous bandwidth will be allocated to communicate the audio data. The 48 kHz audio will be sent from server to client in isochronous packets. The packet rate will be 8 kHz and the audio data per packet will be 24 bytes.

If the client is the Isochronous Resource Manager, the network will operate without difficulty provided the timing in the client is designed properly. The clock for the D/A converters and the clock for the isochronous counter (24.576 MHz) must be derived from the same crystal oscillator. If this is true, the 8 kHz ischronous frame rate is exactly 6 times the 48 kHz-sample rate of the D/A converter. If the server puts 6 stereo audio samples (24 bytes) in each isochronous packet, the D/A converter receives exactly the right number of samples per second. If the client (PC with sound card) is not the Isochronous Resource Manager, problems will occur. If the server is the manager, the 8 kHz isochronous frame will be generated by its crystal oscillator. If this crystal is operating at the high end of its frequency tolerance (+100 PPM), the frame rate will be 8.0008 kHz. The server will put 6 stereo audio samples in each isochronous packet, so the number of stereo samples the client receives is 6 x 8.0008 kHz or 48.0048 kHz.
If the crystal oscillator on the sound card is operating at the low end of its frequency tolerance (-100 PPM), it will accept 47.9952K samples/second. This means the sound card gets 9.6 extra samples per second. If these samples are simply discarded, you will hear a very audible clicking sound.

A possible solution is to use the FIFO and handshaking procedure used in the Fast Ethernet example. When the FIFO in the sound card is nearly full, the client can send a message to server to send an empty packet. Since the server produces 9.6 extra samples per second, on average it would send an empty packet slightly more than 1.5 times per second. Although this method works, it requires additional hardware and/or software to implement. In other words, it costs more.
Consumer Audio Example

Consider another example of a 3-node network consisting of an HMI, a DVD player and a digital amplifier. Assume the HMI node is the Isochronous Resource Manager. Also, assume the DVD drive is decompressing the audio from disk and sending it over the network to the digital amplifier.

The DVD player will have a crystal oscillator and will nominally produce 48K stereo audio samples per second. If the crystal is operating at +100 PPM, the DVD player will generate 48004.8 stereo audio samples per second. If the HMI node has a crystal operating at exactly the nominal frequency, the isochronous frame rate will be exactly 8 kHz. Each isochronous packet from the DVD player will then contain an average of 48004.8 / 8000 or 6.0006 stereo samples/packet. This can be accomplished in different ways.

The DVD drive can send packets with seven stereo audio samples and periodically send an empty packet. In this case, six packets with seven samples will be followed by an empty packet. Once every 1.3 seconds, five packets with seven samples will be followed by an empty packet. This will produce a throughput of exactly 48.0048k samples per second.

Alternatively, the DVD drive can send most packets with six samples (24 bytes). However, occasionally a packet will contain seven samples (28 bytes). To be explicit, every 8000/4.8 (~1667) frames will contain seven samples. Although both of these procedures will put the proper number of samples per second onto the network, controlling the different packet sizes requires some effort and cost.

The digital amplifier which is receiving the audio data from the DVD player has its own crystal which is clocking the D/A converters. If this crystal is operating at –100ppm, then the D/A converter expects to receive an average of 47,995.2 stereo samples per second. Since the DVD player is producing 48,004.8 samples per second the amplifier must do something with the extra 9.6 samples each second. As mentioned previously, if these samples are simply discarded, an untrained listener will hear clicking sound.

Sample Rate Conversion

A solution to the problem in the digital amplifier described above is to convert the sample rate of the data received by the digital amplifier from 48004.8 HZ to 47,995.2 HZ. Sample rate conversion is a well-known process. However, some sort of a digital signal processor is required. This adds cost and can affect quality by producing aliasing artifacts. Sample rate conversion is significantly easier if there is a known frequency relationship between the two rates. In this example, the difference in the sample rates is unknown and must be determined by a processor in the digital amplifier.

Phase Locked Loop (PLL)

An alternative solution is slaving both the DVD player and the D/A converters to the 8 kHz isochronous frame rate. The isochronous resource manager sends synchronization messages every time it's modulo 3072 counter wraps around. The counter in the DVD player will sometimes reach 3072 when it receives the synch message and will sometimes have rolled back to 1. The MSB of this counter will have exactly the same frequency as the Isochronous Resource Manager (8.00000 kHz). However, the Least Significant Bit (LSB) will be jittered.

This jittered 8 kHz signal can go through a phase locked loop to increase the frequency by some integer factor and attenuate the jitter. This higher frequency clock can be used by the DVD drive to read data off the disk at exactly 48 kHz (instead of 48.0048 kHz). Likewise, the 8 kHz signal from the isochronous counter in the digital amplifier can go through a phase locked loop to create a 48 kHz-sample rate clock for the D/A converters. Since both the source and the destination are locked to the same timing source (isochronous frame rate), there is no need for sample rate conversion.

The issue with slaving the DVD player and the digital amplifier to the isochronous frame rate clock is complexity. The jitter on the 8 kHz clocks in the DVD player and amplifier can have a frequency anywhere between 4 kHz and DC. The jitter frequency is directly related to the frequency difference between the crystals in the DVD player (or amplifier) and the Isochronous Resource Manager. Low frequency jitter is possible to attenuate. However, very low bandwidth PLL is required. Typically, low bandwidth PLL has long lock times which may or may not be acceptable. Tricks can be played to reduce bandwidth and keep lock times to reasonable levels. However, they can easily become very complicated.

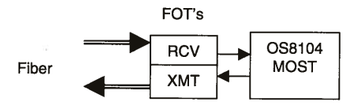
An additional problem with slaving the devices to the frame rate is that each node now has at least two asynchronous clocks. One clock is required by the 1394 interface; i.e. from a crystal. The other clock is the output of the PLL. If care is not taken in designing the digital amplifier circuitry, an aliased or mixed component of the 1394 interface clock will be heard through the speakers.

MOST

MOST (Media Oriented System Transport) is a functional specification developed specifically for interconnecting electronic devices to enable them to share electronic media, whether it is audio, video, voice or data. It defines a basic framework and provides the transport mechanisms to move media efficiently. MOST includes a hardware specification and defines the higher-level system services that need to be provided so unrelated systems can communicate with each other.

MOST Physical Layer

The physical layer is optical fiber. Each node in a MOST network has two optical ports. One for receiving data and one for transmitting. Although the transmission media can be glass fiber, it is typically plastic fiber due to lower cost and ease of installation. Fiber optic transceivers (FOT's) are available from a variety of manufacturers. The MOST network transceiver chip interfaces with the FOT units.



The physical layer of MOST is similar to that of Fast Ethernet since nodes in both technologies have separate transmit and receive ports. Data on the transmission lines for both MOST and Fast Ethernet goes in one direction. In contrast, data on a 1394 transmission line can go in both directions.

A significant difference between the physical layer of MOST and Fast Ethernet is that data is always being sent through the transmit and receive ports of a MOST node. A Fast Ethernet node only transmits data when it is sending a message. If it is not sending a message, its transmission line to the repeater (or switch) will be idle. A MOST node always bypasses the data it receives unless it is generating data. In this case, the node will multiplex its own data with the data that it receives. If no nodes are generating data, idle codes are transmitted through the network.

In a MOST network, one node is defined as the timing master. It has a crystal oscillator, which provides the timing reference for the entire network. It transmits data whether or not any node in the network is trying to communicate. If there is no useful information on the bus

the timing master, simply sends idle codes. All other nodes in the network have phase lock loops (PLLs) which recover their master clocks from the bitstream sent by the timing master.

MOST Topology

Ring

The most effective implementation of a MOST network is a ring. The ring starts with the timing master. Its optical output is connected to the optical input of the next node. The optical output of that node is connected to the optical input of the next node, and so on, until the optical output of the last node is connected to the input of the timing master.

The timing master has a crystal oscillator which generates the timing of its transmitted data. Every other node in the network (slave nodes) uses an analog PLL to recover a clock from the received data. Each slave node simply bypasses its received data if it is not interested in sending any data. When a slave node does input data to the network, it reads in the received data, multiplexes it with data it wishes to communicate, and sends the multiplexed datastream out its transmitter.

Every node in a MOST network operates at exactly the same frequency. This is fundamentally different from FAST Ethernet and 1394.

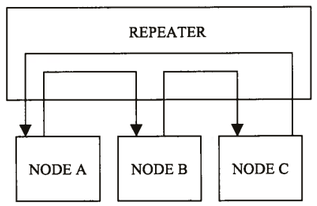


Figure 2.0: MOST with Repeater

Star

A MOST network can be implemented in a star topology (see Figure 2.0) similar to that of FAST Ethernet. A repeater box can be used for each node to connect its transmit and receive ports to. The repeater has little intelligence. If a node is not connected to a port, it is disabled. The repeater box adds cost to the system. However, it does provide fault tolerance. If one node fails, the repeater can detect this condition and bypass the faulty node.

Tree

Like IEEE 1394, a MOST network can be implemented in a daisy chain or a tree topology (see Figure 3.0). Each node can have two sets of transceivers, which enable daisy chaining. The following diagram illustrates how this works. Each node would contain a single transceiver. If a port (RX/TX pair) is not connected (nodes A and C) the transceivers that are being used are connected to the transceiver chip. If both sets are being used (node B), one pair bypasses. The logical topology is still a ring. However, physical topology is a daisy chain.

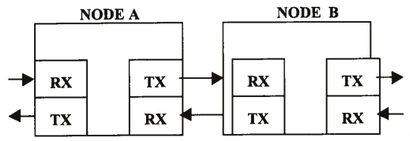


Figure 3.0: MOST with Tree Topology

If three or more transceiver pairs are implemented on a single node, a tree with multiple branches can be implemented. The daisy chain or tree topology is more expensive than a ring since the number of FOT units increase. If desired, it can be done.

MOST Communication Mechanisms

There are two different types of communication mechanisms in a MOST network. They are synchronous and packet. Both Ethernet and 1394 only support packet. Although 1394 provides a variety of packet types including asynchronous and isochronous, they are still packets. MOST provides two types of packet communication channels. They are called asynchronous and control frame. Asynchronous is functionally identical to Ethernet and can support 10 Mbit Ethernet running TCP/IP. The control channel is a unique asynchronous channel allocated its own guaranteed bandwidth.

MOST Frame Structure

MOST has a frame structure similar to a frame in telecommunications. When a long-distance telephone call is made, a number of switches are set up in central offices across the country through which the digitized voice will pass. If all the bandwidth of the network is consumed by other telephone calls, the local central office will inform the caller that all circuits are busy; try again later. If there is bandwidth available, the call is connected and communications can begin. The connection is actually 2 connections; one in each direction. These connections are time slots in a frame that repeats at 8KHz. Voice is digitized at this rate.

A MOST frame repeats at the audio sample rate since the desired audio quality is better than that of the telephone system. The audio sample rate is typically 44.1 kHz or 48 kHz. This repeating frame is divided into 64-byte wide channels. The total bandwidth is 64x8x48KHz = 24.5 Mbit/sec. Of these 64 channels, 2 are allocated to the control channel, 2 are used for the network management functions and 60 are divided between synchronous and asynchronous channels. The boundary between the two is user defined. The useable bandwidth is 62x8x48K = 23.8 Mbit/sec.

MOST Synchronous Channels

The synchronous (sync) channels are used to communicate streaming video, audio and voice data. As in the telephone system, this data flows through the MOST network. It is not packetized, so there is no need for buffers, packet headers or for intelligence at every node to manage them.

When a CD player wants to start playing, it asks the network for bandwidth. If synchronous bandwidth is available (all circuits are not busy), the network will allocate 4 bytes to the CD player.

Each node in the MOST network acts like a central office in the telephone system. A central office is essentially a big cross point switch which routes input voice channels to output voice channels. When a telephone is dialed, a switch is set up which stays in place until the call is ended.

Likewise, a MOST node can route the synchronous data it receives from the network back out on the network or it can route the data off the network. For instance, a power amplifier could route data from the CD player off the network and to its D/A converters. The rest of the synchronous channels could simply be routed back onto the network, digitized voice from a microphone could bypass the amplifier on its way to the cell telephone.

The approach used in MOST to communicate streaming audio, video and voice data is inherently more efficient than the packet-based method in 1394 and Fast Ethernet. The synchronous approach in MOST has zero overhead or 100% efficiency. This is in comparison to the 40% efficiency for audio and 2% efficiency for voice for 1394. These efficiency numbers explain why MOST can communicate 15 stereo audio channels over the 22 Mbit usable bandwidth while 1394 can support a maximum of 20 stereo audio channels with 100 Mbit.

When the data rate of the stream decreases, MOST throughput becomes even greater than 1394, even though the raw data rate is 4 times higher. For example, 1394 can support a maximum of 24 voice channels while MOST can support at least 60. With additional hardware next to the MOST transceiver, 360 voice channels can be supported. These differences are dramatic.

The synchronous nature of MOST eliminates the problems illustrated in the previous 1394 examples of a CD player and digital amplifier. The timing of every MOST node is slaved to the master so they all operate at exactly the same frequency. The CD player will produce samples at the rate determined by the crystal in the master and the D/A converter in the digital amplifier is accepting samples at this rate as well. There is no need for the sample rate conversion or complicated PLLs. This results in higher quality and much lower cost per node.

MOST Control Channel

The control channel is used to send short messages to devices to control various functions. Examples of such messages are to change the track on a CD player or change the volume in an amplifier. The MOST specification defines a complete set of commands, or application protocols, which can be communicated over the control channel. The specification provides enough flexibility and expansion to accommodate common devices as well as unforeseen devices that may be required in the future. Space is also reserved for user-defined devices.

MOST Application Protocols

The MOST application protocols do not have to be used. However, they were developed in cooperation with major auto manufacturers and consumer equipment suppliers specifically for networking consumer devices. This does not exist in 1394 or FAST Ethernet.

Fast Ethernet and 1394 define the mechanisms for sending and receiving data; however, there is no specification for what the data means. Typically, TCP/IP is run over Fast Ethernet and standard applications, such as FTP, sendmail, etc., are provided with the operating systems. These and other applications define the messages in most computer networks.

IEEE 1394 is different. The standard simply specifies an address space. Part of the address space specifies the address of the node. The other part of the address space specifies memory locations inside a node. When messages are sent to a particular node, the data in the message is written to the specified memory locations in

that node. Some of this address space is defined in order to operate the network; however, the majority of the address space is undefined.

Some companies, such as Yamaha (M-LAN), have developed sets of protocols for audio applications. However, they are not part of the specification. This results in physical compatibility between 1394 devices but not software compatibility. Devices from different companies speak different languages.

As defined by the 1394 specification, an isochronous packet has a header, checksum and useful data. The M-LAN specification defines an additional header that goes into the useful data field. This overhead was not considered in the 1394 efficiency calculations described earlier.

MOST Bandwidth

The control channel in MOST consists of two bytes per frame. At a frame rate of 48 kHz this provides 768K bits of bandwidth. Messages can be sent once every 16 frames (block boundary). This means 3000 control messages can be sent per second independent of how much synchronous bandwidth is being used.

A MOST control message consists of a header, user data and a checksum. The user data field can be up to 17 bytes and the efficiency is 53%. As shown in the section on 1394, the total time for a 17 byte message in a large network (16 nodes) running at 100 Mbit is about 1.4 µsec for arbitration, 3.5 µsec to send the message (44bytes * 8bits/byte * 10ns), and 5.6 µsec for acknowledge and sub-action gap. The total time is about 10.5 µsec. This means the efficiency is about (17bytes * 8 bits * 10ns) / 10.5 µsec X 100% or 13%, which is 4 times worse than MOST.

MOST Arbitration

The arbitration mechanism for control frame messages is very similar to that of 1394. Nodes closer to the root node (in 1394) have a higher natural priority while nodes closer to the timing master have a higher natural priority in MOST. In both 1394 and MOST (control channel) one node always wins arbitration and sends a message, which is in contrast to the collision detection mechanism in Ethernet.

IEEE 1394 defines a fairness interval during which all nodes have an opportunity to send one message. The MOST control channel assigns a priority to all nodes (in addition to the natural priority) which decreases each time a node sends a message. Periodically the priority registers are all reset to the highest level. This periodic resetting is analogous to the 1394 fairness interval. These mechanisms are not found in Ethernet.

MOST Acknowledgement

Control frame messages have automatic acknowledgement. If a destination node receives a control frame packet properly, an acknowledge signal is automatically sent back to the transmitting node. If a destination node does not properly receive a packet because its receive buffer was full, the checksum was incorrect, the destination address was incorrect, etc., information is sent back to the sending node specifying why the message was not properly received.

IEEE 1394 provides similar low level acknowledges. However, Fast Ethernet does not. Fast Ethernet expects the higher level protocols to ensure reliable data transmission. If a Fast Ethernet packet is not received properly, the receiving node simply ignores the packet. the transmitting node does not know if the message was received or not. The higher level protocol Transport Control Protocol (TCP) ensures reliable communication. This is done through handshaking messages.

MOST Asynchronous Channel

As a quick review, a MOST frame consists of 64 bytes and repeats at the system sample rate, typically 48 kHz. Of the 64 bytes, two are used for network management, 2 are used for the control channel and the remaining 60 can be divided between synchronous and asynchronous bandwidth. The boundary between synchronous and asynchronous is user defined.

The asynchronous channel forms a packet-based network like the control frame channel. However, no higher level protocols are defined by MOST. It is intended to operate under standard protocols such as TCP/IP. Packets are similar to Fast Ethernet in that they have headers and checksums.

MOST Audio Example

Let's look at the example described in the section on IEEE 1394 of the HMI, DVD player, and digital amplifier nodes. In this case, we assumed the audio from disk was decoded in the DVD player and communicated to the digital amplifier. Since the decoded audio from the DVD player has a sample rate of 48 kHz, the sample rate for the network should be 48 kHz.

Any node in the network can be the timing master. The timing master has a crystal oscillator that generates the timing reference for all other nodes. Every node operates at exactly the same frequency. If the HMI node is the timing master, the DVD player and the digital amplifier slave their timing to the clock recovered from the received MOST bit-stream. The recovered clock from the MOST transceiver chip drives the timing for all other circuitry on each of those nodes.

In the DVD player, the clock recovered from the MOST bit-stream (the output from the MOST transceiver) provides the master clock to the drive mechanism. Therefore data is read from the disk at a rate proportional to the 48 kHz MOST sample rate. The audio data from the decoder also has exactly the same 48 kHz-sample rate as the network. The audio data simply streams into the serial ports on the transceiver chip and onto the network.

The timing in the digital amplifier is derived from the MOST network as well. The D/A converters are operating at exactly the same audio sample rate as the data from the DVD decoder chip. The 48 kHz audio data from the DVD drive is streamed off the network through the serial ports on the MOST transceiver chip and into the D/A converters. There is no need for buffering, creating packets, converting sample rates, building complicated PLL, etc. MOST provides the simplest means to communicate streaming data.

CONCLUSION

The most cost-effective and optimum method for in-vehicle transporting of digital audio and video is synchronous transport. Asynchronous transmission is optimized for routing data, command and control. There are two digital audio interconnection standards: Audio Engineering Society (AES) and Sony-Philips Digital Interface (SPDIF). However, both of these standards are synchronous in nature and call for less than 0.25 of pulse duration of jitter. Control of jitter and delays of asynchronous transport may require proprietary complicated software solutions and additional silicon. Julian Dunn points out in his AES 106 May 1999 issue, "Sample Clock Jitter and Real-time Audio Over the IEEE 1394 High Performance Serial Bus", that CD quality audio is not realizable with current IEEE 1394 specification.

REFERENCES

1. Anderson, "FIRE WIRE SYSTEM ARCHITECTURE", PC Architecture Series.
2. Martin, Local Area Networks Architecture and Implementations, Prentice Hall
3. Watkinson, "The Art of DIGITAL AUDIO", Second Edition, Focal Press.
4. www.mostcooperation.com, login ID is required
5. Laubscher, "AN INVESTIGATION INTO THE USE OF IEEE 1394 FOR AUDIO AND CONTROL DATA DISTRIBUTAION IN MUSIC STUDIO ENVIRONMENTS", HTTP://www.RHODES.AC.ZA, Page

Product/ Technology	Medium	Topology	Max. Number of Nodes		Distance (m)	Speed		
			Physical limitation (per segment)	Address-Space (Bit)		Bit Rate (Mbit/s)	Gross Rate (Mbit/s)	Net Stream Data Rate (Mbit/s)
HiQOS	POF	ring or star with passive star	60 2	60	30 10	54 108 216	36.5 44.9	32 39
IEEE1394 "FireWire"	shielded cable/POF/UTP	Tree	63	63	4.5	100 200 400 800 1600 3200	98.3 196.6 393.2 786.4 1,572.9 3,145.7	<65.5 <131 <262 <524 <1048 <2096
MOST	POF, UTP	Ring, active start, or tree	64	16	100	25 50 150	24.6 49.1 147.5	23.1 46.0 138.3
DC-BUS	DC Line Cable	Bus	16	16	12	0.562 12	0.225 6.8	0.21 6
MML	POF	Star with Coupler	10 (100 with ADN)	156	4 to star	110592	98.304	92.328 73.728 46.464
D2B	POF, UTP	Ring	64	12	100	5.6	5.6	4.2

Table 2.0: A Protocol Comparison Chart

2000-01-C029

What is a Car? Beyond Electronics.

Gian Luigi Longinotti-Buitoni
Ferrari North America

Betty Lou McClanahan
Ryan C.C. Chin
MIT

Merri Ferrell
Carriage Museums at Stony Brook

ABSTRACT

A car today presents itself under many guises: as a "office on wheels," a "family space," a "toy," a "sports car," a go-anywhere vehicle, or some hybrid combination. What it means to "drive" consequently is also changing to include more and more secondary activities over and above the primary activity of "vehicle control." As the car continues to evolve, electronics plays a large role, particularly in the development of secondary activities such as entertainment and communications, and as mechanical functions are gradually replaced by electronics. Nevertheless, despite the obvious extension of the functionality of the vehicle, and its continuing improvements, there is a growing concern today that the passenger vehicle may be losing emotional resonance with the customer. The fear is that it might simply become a commodified "wheeled conveyance" or even an "appliance," despite all efforts to increase its functionality and usefulness.

Under these circumstances, one may certainly ask the question: "What, indeed, is the true nature of "car-ness that we might elevate and protect it?" Or "After all, what IS a car?"

In this paper, we pose an answer to the question "What is a car?" after leafing through its history. We claim that the principal attribute of a "car" as opposed to a "wheeled conveyance" is its enduring ability to emotionally engage the driver, either driving the car, or viewing the exterior. This we claim is the "creative distance" of the car and the driver, a "distance" created by the fact that the car poses significant challenge, emotionally or intellectually, or even physically, to the driver or owner.

We then suggest a theory of product content that will focus industry effort on what we claim is the basis of creative distance, what we call the "firm-specific content." The final section of the paper details the special role that electronics can play in building a creative distance.

WHAT IS A CAR?

Defining the word "car" is somewhat akin to defining the word "spoon" or "fork." One doesn't see the need unless a serious problem of misunderstanding is involved. Perhaps that moment has arrived. Useful definitions of the term simply do not abound in the industry. [1]

But the word "car" has proved useful from at least the 16th century where we find, for example, Shakepeare using it in *Richard the Third:*

"The weary Sun..by the bright Tract of his fiery carre" [2]

The earliest "car," as we know it, was initially thought of as a replacement for the exclusive, highly-designed and constructed private carriages which were luxury items. It certainly could not compete at all with mass-produced buggy which was sold for as little as $25. But it assumed the names of carriages. Cars then were designated by terms like "tonneau," "buggy," or "runabout."

Search

Search Results

CRITERIA

Sort by: Relevance Published Title

Text: "international congress on transportation electronics"

Date: 2000

RESULTS

Display: Grid List

Viewing 1 to 30 of 73

Development of the Hybrid Vehicle and Its Future Expectation

2000-11-01 Technical Paper

Shinichi Abe **2000-01-C042**

Following its introduction into the Japanese market, the Prius, Toyota Motor Corporation's hybrid vehicle, was released onto the American and European markets in Mid-2000. This lecture will take the form of an explanation of the new technology used to meet the demands of the western market, including improved driving performance, lower emissions, down-sizing of the system and lower costs. The lecture will also look briefly at the new Toyota hybrid system, the THS-C, which is currently being developed. The talk will include a look at the future of the hybrid vehicle.

JamaS Study on the Location of In-Vehicle Displays

2000-11-01 Technical Paper

Tsutomu Asoh, Kenji Kimura, Toshiyuki Ito **2000-01-C010**

JAMA (Japan Automobile Manufactures Association, Inc.)'s guideline for car navigation systems is being decided on displayed the amount of information while driving. The position of a display and the estimated equation, which could be applied from a passenger car to a heavy truck, was studied. The evaluation index was the distance which drivers could become aware of a preceding vehicle by their peripheral vision, because car accidents while drivers glance at an in- vehicle display are almost the rear end collisions. As the results, the lower limit of a position of an in-vehicle display for a passenger car was 30 degrees, and a heavy truck was 46 degrees.

Automotive Electronics-A Challenge for Systems Engineering

2000-11-01 Technical Paper

Maximilian Fuchs, Jürgen Ehret **2000-01-C048**

This paper presents the challenges in automotive electronics. Considering the deficiencies of the current ECU (electronic control unit) design process, a new design process is outlined. This design process mainly focuses on the independence of the ECU hardware architecture development and the software function development.

Evolution of Automotive Semiconductor Pressure Sensors

2000-11-01 Technical Paper

Iwao Yokomori, Tiaki Mizuno **2000-01-C054**

Recently, there is a need for new applications of pressure sensor, such as direct fuel injection systems for protecting the environment, or power assisted brake systems for improved driving safety. For these widening areas of application, pressure sensors with higher accuracy, a wider pressure-sensing range, and integration of sensor chip functions are required. This paper discusses our development of automotive semiconductor pressure sensors.

The Future of Radio Is Clear: Satellite Radio

2000-11-01 Technical Paper

Michael J. Bergman **2000-01-C026**

This paper gives an overview of Satellite Digital Audio Radio Service (SDARS), discusses some of the benefits and technical challenges, and introduces some next-generation features of SDARS.

Role of Electronics in Automotive Safety

2000-11-01 Technical Paper

Priya Prasad **2000-01-C086**

The past, current and future role of electronics in reducing accidents, crash severity and crash notification is discussed. A holistic approach that ties pre-crash, crash and post-crash factors in enhancing automotive safety is examined and the growing role of electronics in affecting the three factors is discussed. Electronic technology has already entered the automotive safety arena, and its utilization in the future is expected to grow rapidly towards the goal of safer roadway environment.

Sensor Vision and Collision Warning Systems

2000-11-01 Technical Paper

Ulrich Seger, Peter M. Knoll, Christoph Stiller **2000-01-C001**

Due to an earlier analysis of the interrelation between collisions and advanced driver reaction a significant number of accidents could be avoided through timely threat recognition and appropriate maneuvers for collision avoidance. This may be achieved either by suitable warning to the driver or by automatic support to longitudinal or lateral control of the vehicle. A precondition for the registration of the dangerous situation is the incorporation of appropriate sensors. This leads to an surround sensor vision system accompanied by a matched human machine interface. Many vehicles readily offer ultrasonic reversing aids as add-on systems. Furthermore, long-range radar systems for adaptive cruise control are now coming on the market. New sensor technologies, such as short-range radar and video, which are currently under development, open up a plurality of novel functions thus enhancing driving safety and comfort.

Safety, Mobility and the Environment: the Electronic Cocoon

2000-11-01 Technical Paper

Frode Maaseidvaag, Paul Mulvanny, K. Vankatesh Prasad **2000-01-C003**

An electronic cocoon is a vehicle in which the requirements of safety, mobility and the environment are managed electronically using information from on-board and off-board data networks. The center of these data networks is the driver's own complex and adaptive network which has interfaces of its own with sensory, cognitive and motor capabilities and are variable across the population. In this paper, we describe the delicate balance between the unbounded desires for the ideal and the viable expectations of reality. Our vision will be described in the light of significant and exciting technological advances, sobered by the realities of our increasingly complex driving environment --- an environment that actively and constantly challenges the driver's finite attentive resources.

The Emerging Fabric of Seamless Mobility

2000-11-01 Technical Paper

K. Venkatesh Prasad **2000-01-C033**

Consider this recent data about the "speed" of the information age: In a sixty-month window beginning in the year 1995, the number of cellular phone subscribers increased from 90 million to 330 million, the number of people on the internet went from about 5 million to just under 200 million, and not surprisingly the number of websites grew from a few thousands to 50 million. Now further consider that almost every cellular phone subscriber is also an automobile user, be it as a driver or a passenger. A fabric of mobility soon emerges --- one that is formed by the meshing of the strands of information mobility with those of personal mobility. In this paper we explore a "new frontier'' --- a "territory'' on this multidimensional fabric that is defined by the demands of seamless mobility. The typical daily cycle of events in our lives involve many different modalities and time-phases of mobility. We examine the rapidly increasing demands of seamless mobility in this context.

Development of a More Efficient and Higher Power Generation Technology for Future Electrical Systems

2000-11-01 **Technical Paper**

Hiroaki Ishikawa, Atsushi Umeda, Masatoshi Kohmura **2000-01-C081**

Segment conductor technology can reduce the resistance of the stator winding by 50%, and noise sources can be cancelled through the dual winding and dual rectifier arrangement. This technology provides the generator with not only high power and efficiency but also low acoustic noise and electrical noise. This technology, combined with the switching rectifier technology, will be adopted into all charging systems including the conventional type, the high voltage type, and starting/engine-assist type.

Open Systems and Open Mindsets: Entertainment, Information and Communication Systems for the Automotive Future

2000-11-01 **Technical Paper**

Rolf Juergen Bruess **2000-01-C085**

This paper and presentation will describe how one company is addressing the convergence of entertainment, information and communications for tomorrow's automobiles. The approach presents a framework for incorporating diverse technologies in cost-effective and efficient ways to provide for both flexibility and future growth. It offers a paradigm for integrating best-in- class technologies from third parties into multimedia systems that meet fast- changing customer demands. The approach recognizes that automotive customers' expectations for features and functions are increasingly driven by computing and telecommunications capabilities that they experience outside of their cars. Future in-vehicle multimedia systems will be based on combinations of leading edge technologies involving computers, cellular or satellite communications, car radio and car navigation product technologies.

Intelligent Hall Effect-Based Magnetosensors in Modern Vehicle Stability Systems

2000-11-01 **Technical Paper**

V. Gussmann, D. Draxelmayr, J. Reiter, T. Schneider, R. Rettig **2000-01-C058**

After comparing magnetosensor technologies for automotive use the system aspects of wheelspeed sensors for vehicle stability systems are discussed. A new generation of intelligent differential Hall Effect-based sensors is described focussing on technology, operating principle and circuitry of the Hall IC. The final realization of the wheel speed sensor is presented concluding with a summary of the main advantages of this concept.

Circuit Protection Solutions for 42v Automotive Electrical Architectures

2000-11-01 **Technical Paper**

Michael E. Williams **2000-01-C074**

This paper examines the obstacles faced by blade fuses with a single piece, zinc construction; it also introduces a complete line of circuit protection solutions designed for the upcoming 42V PowerNet. These new fuses employ multi- material construction, fiberglass reinforced housings and safety features to prevent interchange of high and low voltage fuses.

Vision of Mobile Information Services

2000-11-01 **Technical Paper**

Tsuneo Shiga **2000-01-C017**

As wireless technologies evolve, in-vehicle information services are becoming more and more essential to vehicle users. In contrast with information services in the home, in-vehicle information services emphasize the use of information to make driving more comfortable, rather than simply displaying information during driving. In particular, traffic information is, unlike other kinds of information, effective in getting to a destination and therefore, must be real-time to be useful. In Japan, car navigation systems have a large market penetration; dynamic route guidance systems (DRGS) operating in concert with navigation systems have been popular since 1995. This paper discusses mobile information services including DRGS. The focus is on the Japanese market where navigation technologies are the most advanced.

Dynamic Route Guidance - Status and Trends

2000-11-01 **Technical Paper**

Andreas Eppinger, Ernst P. Neukirchner **2000-01-C013**

Most of the existing car navigation systems are operat-ing autonomously using a static digital road map on-board. The benefit of such systems lies mainly in trips through un-known street networks. Systems providing routes dynamically adapted to the traffic situation ease driving even in well-known areas. Two principles can be used for dynamic route guidance. One is receiving the description of a complete recommended route from a traffic center. The other one is just receiving encoded traffic messages for updating a static map according to the traffic situation. In the first case, route calculation has to be performed centrally by the service provider, while the second method needs the on-board route calculation. Both methods have advantages and disad-vantages and need a method to refer to their local street map. The status of such systems and trends are described.

Commercial Vehicle Application of Dynamic Route Guidance

2000-11-01 **Technical Paper**

James L. Bander, Chelsea C. White **2000-01-C015**

Once the infrastructure and in-vehicle systems can support dynamic route guidance, commercial vehicles may be among the early adopters. In order to estimate the value of an information system to the commercial vehicle operator, we consider the value of information in the context of a stochastic shortest path problem. In the presence of a dynamic route guidance system, there may be advantage to re-routing in-route based on real-time traffic information. After developing a mathematical model, we solve a numerical example in which dynamic route guidance produced a travel time savings of almost 11%.

FordS Zero Emission P2000 Fuel Cell Vehicle

2000-11-01 **Technical Paper**

Kurt D. Osborne, Mark S. Sulek **2000-01-C046**

The P2000 Fuel Cell Electric Vehicle developed by Ford Motor Company is the first full-performance, full-size passenger fuel cell vehicle in the world. This development process has resulted in a vehicle with performance that matches some of today's vehicles powered by internal combustion engines. The powertrain in Ford's P2000 FCEV lightweight aluminum vehicle consists of an Ecostar electric motor/transaxle and a fuel cell system developed with XCELLSiS-The Fuel Cell Engine Company (formerly dbb Fuel Cell Engines, Inc.). Ballard's Mark 700 series fuel cell stack is a main component in the fuel cell system. To support this new FCEV, Ford has constructed the first North American hydrogen refueling station capable of dispensing gaseous and liquid hydrogen. On-going research and development is progressing to optimize fuel cell vehicle performance and refueling techniques.

Field Operational Test Results of An Automated Collision Notification System

2000-11-01 **Technical Paper**

Joseph Kanianthra, Arthur Carter, Gerard Preziotti **2000-01-C041**

This paper describes a Field Operational Test (FOT) of an Automated Collision Notification (ACN) System, a new way to provide definitive pre- hospital medical care to people injured in motor vehicle crashes. This FOT is part of the National Highway Traffic Safety Administration's (NHTSA's) research being conducted under the U.S.

Department of Transportation's Intelligent Transportation Systems (ITS) program. Management of the program is by the Office of Vehicle Safety Research in NHTSA. The ACN FOT is a demonstration of the application of advanced technology for the improvement of pre-hospital emergency care for motor-vehicle crash victims. The test, involving approximately 850 volunteers' vehicles, was conducted in rural Western New York State. A partnership of local public agencies and private corporations, led by Veridian, Inc., performed the test.

Development of Lead-Free Soldering Technology and Its Application

2000-11-01 **Technical Paper**

Kenichiro Suetsugu **2000-01-C043**

In recent years, the solder used to mount electronic components onto printed circuit boards has become a social issue because of the lead it contains. To resolve the problem, the development of a lead-free solder and bonding technology that utilizes it are not only research themes of industry but national research programs as well. In light of the circumstances, we developed an Sn-Ag solder with minute quantities of Bi and other additives, and applied it for the first time in the industry to a printed circuit board for portable MD player. The developed solder has proven workable under current production temperature conditions (profile that peaks at 230 degrees centigrade) and reliable for bonding electronic component leads in mass-production.

ProductivityS Next Dimension - The Mobile Office Computing Platform

2000-11-01 **Technical Paper**

Dean Kanemitsu **2000-01-C027**

Mobile professionals have longed for a true office on wheels because they are productive everywhere except in the place that they spend the most significant amount of time - the vehicle. With traffic congestion ever increasing, the more time the mobile professional is spending in the vehicle. This lack of productivity in the vehicle results in a significant loss of revenue and income for the mobile professional. This paper examines who the mobile professional is, identifies the requirements of the mobile professional, shows the hardware and software architecture for the in-vehicle portion of the mobile office, and identifies the challenges to overcome to create a true mobile office.

Onboard Hardware & Software: Common Open Standard Solutions

2000-11-01 **Technical Paper**

Scott Andrews **2000-01-C020**

The Automotive Multimedia Interface Collaboration has been developing a unified architecture with supporting networking and software API specifications. The objective of this effort is to simplify the task of developing hardware and software for multiple automotive platforms and makers. This paper will describe the AMIC organization, and provide a summary of the progress to date. The overall architecture requirements will be briefly described and the expected impact of these requirements on applications and products will be analyzed.

Reinventing the Car Radio for the Internet-The IradioSt

2000-11-01 **Technical Paper**

Parvathy Bhaskaran **2000-01-C025**

The car radio has been an integral part of the delivery of information and entertainment to the vehicle for more than half a century. With the advent of the Internet, the nature of the infotainment content available to the consumer has dramatically changed. This paper describes an infotainment device, similar to Motorola's iRadio and hereafter generically referred to as the i-radio, which leverages the familiar car radio paradigm to allow the user to safely "browse" through a variety of Internet and traditional content using multi- modal input and output. It further discusses a robust, secure Java-based framework that enables the rapid development and deployment of distributed infotainment and safety service applications for this device.

Child Occupant Safety - What Might We Expect

2000-11-01 **Technical Paper**

Robert C. Lange **2000-01-C039**

The air bag safety issues became evident in 1995 and other factors have conjoined to change the climate regarding motor vehicle safety. Traditionally, motor vehicle safety issues have been evaluated based upon the effects upon average adult males. The new climate requires consideration of the effects on persons of differing size and gender. By including consideration of children and women, rulemaking and the applied technologies are able to better optimize safety than is the case when rules are focused only on the average adult male. Automotive electronics serves a key role in the migration from a one-size-fits- all protection to a more customized protection for a variety of occupants. The enhancements have been the most prominent in the area of sensing, be it the sensing and characterization of the crash itself, or the sensing and characterization of occupants in the vehicle.

Advancements in Crash Sensing

2000-11-01 **Technical Paper**

Anson Foo, Stephen A. Ridella **2000-01-C036**

The crash modes that occur each day on streets and highways have not changed dramatically over the past 50 years. The need to better understand those crash modes and their relation to rapidly emerging, tailorable restraint systems has intensified recently. The algorithms necessary for predicting a deployment event are based on an approach of coupling the occupant kinematics in a crash to the sensing technology that will activate the restraint system. This paper describes methods of computer modeling, occupant sensing and vehicle crash dynamics to define a crash sensing system that reacts to a complex set of input conditions to invoke an effective restraint response.

Concepts Designed to Enhance the CustomerS Driving Experience

2000-11-01 **Technical Paper**

S. Buckley, K. Johnson **2000-01-C031**

Throughout its history, the automobile has served the utilitarian purpose of transportation quite well. However, until recently, vehicle occupants have had little else to do while proceeding from point "A" to point "B." The phenomenal improvements in computing and communication technologies promise to evolve the driving experience. Like never before, new opportunities to make driving more efficient and engaging are becoming available. The challenge will be to develop the right combination of technology, safety, design and user interface that creates a product popular with customers.

An Open Versus Closed Architecture for Multimedia Systems

2000-11-01 **Technical Paper**

Ralph L. Robinson **2000-01-C065**

For many years, carmakers have developed unique system designs to gain a competitive advantage using some unique technology or an optimization of a design to cut costs or improve quality. This leads to continual increase in complexity, long development times and high development costs. A common platform, based on an "open architecture," provides a solution for many of the problems associated with the conventional automotive approach to electrical/electronic system designs. The PC industry is a prime example of how an open architecture can provide benefits to the consumer, manufacturers of software and hardware components, as well as complete system integrators. The PC, based on the initial IBM computer developed in the early eighties, has become a de facto standard that has survived 20 years of fast and dramatic changes in the fundamental technologies used within the platform.

Power Electronics and Electric Machinery Innovations - U.S. GovernmentS Role in Pngv

2000-11-01 **Technical Paper**

Donald J. Adams **2000-01-C063**

The U.S. Government plays an important role in the Partnership for a New Generation of Vehicles' (PNGV) electrical and electronics technologies with a program consisting of high-risk research and development (R&D) projects. The Department of Energy (DOE) plays the largest role in supporting these technologies to specifically address

automotive needs. DOE has three Automotive Integrated Power Module (AIPM) contractors and two Automotive Electric Motor Drive (AEMD) contractors working to become viable suppliers for PNGV. Materials development projects are working to improve materials and devices needed in automotive motors and drives, such as permanent magnets, capacitors, sensors, connectors, and thermal management materials. Advancements in inverters, controls, and motors and generators conducted at DOE's national laboratories are also presented.

Smart Sensing in Steering and Chassis Applications: a Digital Torque and Absolute Angle Sensor

2000-11-01 **Technical Paper**

David Bernardi, John Baxter, Alex Mortara **2000-01-C057**

This paper describes the operation of an advanced CMOS opto-ASIC used in a digital torque and angle sensor. Its purpose is to resolve the relative and absolute angular position of two bar-coded disks rotationally coupled by a stiff torsion bar. The resulting digital output of the sensor comprises two absolute angle measurements with up to 18-bit accuracy, and a torque measurement calculated from the difference of the two angular measurements. The measurement of torque and absolute angle in a single sensor eliminates the need for additional steering angle sensors for other vehicle dynamic systems such as dynamic stability control (DSC) or intelligent cruise control (ICC).

A Process to Design and Master the Global Vehicle Electronic System Architecture

2000-11-01 **Technical Paper**

Mondher Attia, Olivier Cayrol, Alexis Drogoul, Rémy Foisel, Françoise Simonot **2000-01-C066**

The introduction of the electronic sub-systems in the automotive vehicles and their tight level of interactions have led to an ever-increasing complexity of the embedded global electronic system. In addition to this complexity, the system architect has to deal with diversity of automotive products and to favor the reusability, while being constrained by the difference of visibility he has on each sub-system. Thus, managing the global vehicle electronic system appears to be very hard and necessitates adapted design methods and tools. Nevertheless, it eases the insertion of innovations in the vehicle and permits to differentiate products and adapt them to each customer's needs. This article presents a design method for electronic system architecture - called CAROSSE - that ensures modularity and reactivity and promotes dependability studies and system-level validation. It takes advantage of heterogeneous multi-agent system techniques and modeling techniques in queuing system formalism.

The Development of Traffic Information System Using Autopc

2000-11-01 **Technical Paper**

Yoshiki Chubachi, Hiroyuki Okagaki **2000-01-C016**

AutoPC realizes the flexible design the ITS platform. The system can utilize Win32 based communication, navigation, networking, audio, speech and vehicle application interfaces provided from Microsoft or developed by each OEM. This will ensure a quick and flexible development environment for the system, including the hardware, software, and peripherals. This paper will describe in detail the building blocks concept used in this ITS platform, including examples of potential implementations.

Search

Search Results

CRITERIA

Sort by: Relevance Published Title

Text: "international congress on transportation electronics"

Date: 2000

RESULTS

Display: Grid List

Viewing 31 to 60 of 73

Human-Machine Interface: How to Make It Simple and Effective

2000-11-01 Technical Paper

Peter M. Knoll, Winfried H. Koenig **2000-01-C019**

We are faced with a rapidly increasing flood of information to the driver. In addition to established information systems (car radio, vehicle monitoring, mobile phones), high class vehicles feature navigation systems almost as standard. In the next decade, driver assistance and collision mitigation systems will appear in vehicles. Hence, there is an increasing demand for supplying the driver with more information that help him to drive safer and more economical. In parallel, the price decline in the computer market and the availability of powerful graphic hard- and software concepts make it possible to enhance the classical functions of the instrument board to an interactive multifunctional information panel, and the dashboard will be the main interface between car and driver.

Mobile Information Systems Overview the End-To-End Solution

2000-11-01 Technical Paper

Hasse Johansson, Anders Eliasson **2000-01-C018**

The global market for mobile communication, computing and content (C3) delivery is exploding. Consumers are demanding ubiquitous C3: any content, anytime, anywhere. Examples include smart cell phones with WAP Internet Browsers and "Convergence" portable devices such as the Palm VII Personal Digital Assistant. This paper will explore the special opportunities and challenges as Mobile Information or "C3" is brought into the vehicle world. For example, how will the human-machine interface be designed to allow the driver to "surf the internet"? How will we deploy common, open standards for vehicle hardware and software to reduce cost and product development cycle time? What are the opportunities for future wideband wireless connectivity to vehicles? How will information services be provided? And what kind of media content will be available and when?

Real-Time Traffic Information for Dynamic Route Guidance

2000-11-01 Technical Paper

Larry Sweeney, Joan Ravier **2000-01-C014**

The phrase "Dynamic Route Guidance" has different meanings to different people. This paper discusses various functionalities that can be classified as Traffic-Dependent Route Guidance, their advantages and disadvantages to travelers, their benefits and risks to product providers, their real-time information requirements, and the availability and/or unavailability of such real-time information now and in the future. The paper also discusses the need to evaluate various Dynamic Route Guidance functionalities under actual driving conditions by a variety of drivers in order to understand driver needs and preferences, and in order to assess product risks and liabilities.

Effective Utilization of In-Vehicle Information: Integrating Attractions and Distractions

2000-11-01 Technical Paper

Barry H. Kantowitz **2000-01-C011**

The modern passenger vehicle contains numerous sources of information. In one sense, all of the messages sent from in-vehicle devices are attractive, at least from the viewpoint of the designer who has incorporated them into the vehicle to make driving more pleasurable and safer. Yet in another sense, these same messages can present distractions to the driver resulting in diminished driving pleasure and possibly unsafe vehicle control. Thus, a message that at one moment might be attractive and useful to the driver, at a different moment, especially one where attention must be focused outside the vehicle, becomes an unwanted distraction. This paper reviews three sources of in-vehicle information: advanced traveler information systems, safety and collision avoidance systems, and convenience and entertainment systems. A framework for integrating these sub-systems is outlined based upon human-centered design principles and functional characteristics of systems.

Electronic Cocoon: Product Liability Aspects of Driver Support Systems in Europe and the United States

2000-11-01 Technical Paper

K. A. P. C. van Wees **2000-01-C006**

Especially in the United States, but also in Europe, product liability has often been identified as a possible constraint for the successful implementation of electronic driver support systems. This paper broadly examines product liability law in the United States and in Europe. There is a clear and striking difference in the level of litigation against producers, which is much higher in the United States than in Europe. Briefly assessing the concept of a defective product in both European and American law in relation to electronic driver support systems, it is concluded that the most important differences are probably not caused by diverging substantive product liability law rules but rather by other differences between the legal system in the US and in Europe.

Consumers, Electronics, and the Link to Hybrid Vehicles and the Environment

2000-11-01 Technical Paper

Gary A. Cameron, Richard Lind **2000-01-C045**

The interdependence of consumer features, new electronic and electrical architectures and hybrid propulsion systems are examined. There are two major forces driving future vehicle electronic and electrical systems, one is consumer demand for comfort and safety, and two is the demand for reduced fuel consumption and emissions. These forces are linked by the use of electronics to control vehicle energy generation and usage while providing managed solutions to these demands. Automobile consumer features are discussed and the case is made that these features will require more electric power to be installed on the vehicle. The presence of this increased electric power will then enable the hybrid vehicle functions that will benefit fuel economy and emissions performance.

Vw Lupo, the WorldS First 3-Liter Car

2000-11-01 Technical Paper

Hanno Jelden **2000-01-C044**

After the success of the 4-cylinder 1.9-liter TDI and SDI direct-injection diesel engines in the Passat, Jetta and Polo classes, a new 3-cylinder TDI has been developed for use in the "Lupo 3L,' a compact car with a fuel consumption of 3 liters per 100 km. A new injection system with unit injectors, together with a fully electronically controlled engine management system featuring drive-by-wire- technology, a turbocharger with variable turbine geometry and a fully automated mechanical gearbox and clutch, for the first time ensures the potential to meet the stringent D4 exhaust emissions level and to achieve excellent fuel economy. The wheel-torque based engine and gearbox management systems optimize engine operation in terms of efficiency and emissions.

Infrared Smart Sensors for Climate Control, Person Detection and Air Quality

2000-11-01 **Technical Paper**

K. Jörg Schieferdecker, Jürgen Schilz, Wilhelm Leneke, Marion Simon, Karlheinz Storck, Mischa Schulze, Michael Schrüllkamp, Wolfgang Schmidt **2000-01-C053**

Passive thermal infrared (IR) sensors are commonly used as non-contact temperature sensors, presence detectors or IR absorption gas sensors for many consumers and industrial applications. Due to their decreasing costs, small size, newly integrated signal conditioning and high reliability, these sensors are commencing into several automotive applications. This paper presents new thermal IR-sensors and describes possible automotive applications.

Intelligent Sensing System to Infer DriverS Intention

2000-11-01 **Technical Paper**

Hiroshi Takahashi, Kouichi Kuroda **2000-01-C056**

An approach to designing an intelligent vehicle controller for partially supporting driver operation of a vehicle is proposed. Vehicle behavior is regarded as a system performed by the interaction between the driving environment, vehicle as a machine and driver expectations for the vehicle movements. Driver intention to accelerate or decelerate is mainly generated by the perception of the driving environment. The model we propose involves information on the driving environment affecting driver intention taking driver differences in perceiving the driving environment into account. An engineering model for installing the vehicle controller is expressed by a multipurpose decision-maker allowing explicit treatment of the driving environment, vehicle action, and driver intention. A reasoning engine deals with differences in individual driver traits for generating intention to decelerate by using fuzzy integrals and fuzzy measures.

Design and Implementation of a Dual Processor Platform for Powertrain Systems

2000-11-01 **Technical Paper**

Alberto Ferrari, Sergio Garue, Maurizio Peri, Saverio Pezzini, Luca Valsecchi, Francesco Andretta, Walter Nesci **2000-01-C050**

This paper describes a dual-processor platform for automotive powertrain control with a high-bandwidth interconnection network among processors, memory, and I/O sub-systems, which is suitable for a System-On-Chip (SOC) implementation. The two processors share memory and I/O address space and can operate in parallel at full speed. The cost of this solution, in terms of gates and power dissipation, is not substantially higher than more classical architectures with a multi-master bus or multiple processor busses connected through gateways, but offers almost twice the performance.

Functional Integration of E/E Systems

2000-11-01 **Technical Paper**

Juergen Bortolazzi, Martin Ulrich, Thomas Raith **2000-01-C052**

The complexity of electrical/electronic vehicle systems mandates a systematic approach to the development of vehicle control, infotainment or comfort functions as well as the integration of these functions in an in-vehicle network consisting of several dedicated bus systems and according gateways. Due to reduced time-to-market, the integration has to be performed in a virtual environment. The classical Digital Mockup (DMU) addresses the physical integration of EE systems as mechanical components. However, functional aspects play a dominant role in EE vehicle systems. For this reason, functional integration defines a multi-view, mixed-level approach to the description, transformation, verification and integration of vehicle functions under consideration of the physical vehicle integration.

What Is a Car? Beyond Electronics

2000-11-01 **Technical Paper**

Gian Luigi Longinotti-Buitoni, Betty Lou McClanahan, Ryan C. C. Chin, Merri Ferrell **2000-01-C029**

A car today presents itself under many guises: as an "office on wheels,'' a "family space,'' a "toy,'' a "sports car,'' a go-anywhere vehicle, or some hybrid combination. What it means to "drive'' consequently is also changing to include more and more secondary activities over and above the primary activity of "vehicle control.'' As the car continues to evolve, electronics plays a large role, particularly in the development of secondary activities such as entertainment and communications, and as mechanical functions are gradually replaced by electronics. Nevertheless, despite the obvious extension of the functionality of the vehicle, and its continuing improvements, there is a growing concern today that the passenger vehicle may be losing emotional resonance with the customer. The fear is that it might simply become a commodified "wheeled conveyance'' or even an "appliance,'' despite all efforts to increase its functionality and usefulness.

Challenges in Simulation and Sensor Development for Occupant Protection in Rollover Accidents

2000-11-01 **Technical Paper**

Madana Gopal, Ed Wallner, Stephen N. Rohr, Mary Fitch **2000-01-C038**

Automotive occupant safety continues to evolve. At present this area has gathered a strong consumer interest which the vehicle manufacturers are tapping into with the introduction of many new safety technologies. Initially, individual passive devices and features such as seatbelts, knee- bolsters, structural crush zones, airbags etc., were developed for to help save lives and minimize injuries in accidents. Over the years, preventive measures such as improving visibility, headlights, windshield wipers, tire traction etc., were deployed to help reduce the probability of getting into an accident. With tremendous new research and improvements in electronics, we are at the stage of helping to actively avoid accidents in certain situations as well as providing increased protection to vehicle occupants and pedestrians.

Perceived Quality

2000-11-01 **Technical Paper**

Michael Vernon Robinson **2000-01-C030**

Quality has long been one of the major competitive advantages in industrial products such as automobiles. But build quality has become mandatory across the automotive field, and manufacturers now search for new competitive advantages. An underestimated advantage in the automotive world, which applies in fact to all producers of consumer goods and services, lies in what clients perceive as quality and not what the factory or R&D department considers lack-of-defects build quality. The very definition of quality therefore is evolving. The difficulty with the implementation of "Perceived Quality'' in the engineering- oriented automobile industry is an intrinsic, cultural reluctance to accept apparently unmeasurable, subjective evaluation methods as opposed to those widely recognized, seemingly objective methods used today. This paper will compare present quality evaluation methods with the new perceptual approach.

The Invisible Hand: How to Design the Best Interaction Between Advanced Driver Assistance Systems and Car Users

2000-11-01 **Technical Paper**

Paola Carrea, Enrica Deregibus, Roberto Montanari **2000-01-C034**

In this paper, the invisible hand concept for the definition of Advanced Driver Assistant Systems (ADAS) Human Machine Interfaces (HMI) is introduced. The main idea is that these systems become a sort of "invisible hand'' able to support the driver and to increase driving safety if they are integrated with each other (i.e., longitudinal, lateral and rear control) and the most "persuasive'' and "efficient'' interaction between driver and ADA systems is developed. Here the analysis is conducted taking into consideration the technological evolution scenario and the identification of related user interaction aspects (including technical and communication analysis). Moreover, the ways to design the best interaction between drivers and ADAS interfaces from communication and HMI points of view has been included as well as an overview of the user centered approach adopted in the Centro Ricerche Fiat's design process.

Development of the Motor-Assist System for the Hybrid Automobile--The Insight Development of the Motor-Assist System for a Hybrid Car--Insight

2000-11-01 **Technical Paper**

Kenji Nakano, Shinobu Ochiai **2000-01-C079**

A motor-assist system has been developed and employed for the "Insight' hybrid car. The system consists of an internal combustion engine as the primary power source, with an electric motor placed around the engine's crankshaft. Such construction reduces the system's volume significantly and offers more flexibility for the power plant layout. The system's functions include regeneration during braking, an idle stop mechanism, driving power assistance, and power supply for the 12V electrical system. A proper energy management method for various driving modes has been established by combining these functions, and fuel economy is significantly improved as a result. As another control feature, an active motor vibration control system compensates the idling vibration that is unique to three-cylinder engines.

Dynamic System Interaction in the 42 V Power Network

2000-11-01 **Technical Paper**

Christian Jung, Joachim Melbert, Achim Koch **2000-01-C082**

New high-power loads like Electromagnetic Valve Train (EVT) or "X-by-wire" systems are the driving forces for the introduction of the 42 V Power Network. These systems show high average power consumption and can cause extreme power transients, which result into a substantial variation of the supply voltage. Such distortions deteriorate the performance of the individual electrical and electronic systems in the car and can stimulate critical interactions between them. An analysis of the system units and the complete network, based on simulations and measurements, must ensure the functionality under all operating conditions. This analysis was performed for a system containing an EVT system, a crankshaft starter-generator, a 36 V battery and an additional energy storage element, together with the wiring harness, and further electrical loads.

The Memory Stick in Mobile Entertainment

2000-11-01 **Technical Paper**

Tadao Fujimoto **2000-01-C078**

As semiconductor technology advances, the flash memory is becoming pervasive in high gear. Expectations of flash memory as a recording medium for next- generation digital devices are high, for its field of application is wide-ranged including digital still cameras, portable audio devices, mobile phones, and IC recorders. In such context, Sony has developed an IC recording medium called "Memory Stick," to realize a world where digital devices can be literally "connected." The Memory Stick is not only a medium for exchanging data among audiovisual devices and IT devices, but also aims to be the de facto standard of recording media for EMD (Electronic Music Distribution) and data distribution. Memory Stick will do this by combining a highly efficient audio compression algorithm with copyright-protection technology, both developed by Sony.

Speech Processing for Vehicles: Whats Next?

2000-11-01 **Technical Paper**

Th. Kuhn, R. Theis **2000-01-C077**

A human voice is and remains the best form of communication. Therefore, spoken language interfaces to computers are a topic that has engaged engineers and speech scientists since the fifties. Once limited to the realm of science fiction, speech technology has now passed the threshold of practicality. The commercial deployment of these systems has already begun, although the applications are still specialized for certain purposes. This paper presents a speech recognition system that is particularly suited for drivers and passengers in a vehicle environment. The first chapter is describing a typical system design of today. Tens of thousands of these units are already in use. Thereafter a future system design will be described, in which the voice control of a navigation system is the most challenging problem. Finally the papers shows as an outlook, what comes next after the future. In particular, distributed speech processing systems will be used based on client-server architecture.

Dependable E/E System Drivers and Application Issues

2000-11-01 **Technical Paper**

Brian T. Murray, Robert E. Steele, Horst Schubotz **2000-01-C064**

Today, electrical/electronic systems like ABS/power brakes and electric power steering are all designed to enhance, not replace a mechanical function. If an electrical or electronic fault occurs, the function reverts to the base mechanical capability. Future E/E systems, such as steer-by-wire and brake-by- wire replace mechanical linkages with electrical or optical signals as in computer networks. While these systems offer many potential safety benefits, they will require different strategies for dependability, and as with any vehicle system, they will further require that dependability be an integral part of the overall E/E system design. This paper illustrates how by-wire systems drive different dependability requirements and discusses some key technologies that are emerging to meet these requirements.

Energy Management in DaimlerchryslerS Pngv Concept Vehicle

2000-11-01 **Technical Paper**

Robert L. Davis, Thomas L. Kizer **2000-01-C062**

DaimlerChrysler's PNGV Concept Vehicle represents the fruits of a marriage between DaimlerChrysler's proprietary efforts in developing advanced vehicle body and chassis technologies applicable to PNGV goals and a mild hybrid electric (Mybrid) powertrain developed under government contract. The powertrain was developed as a part of DaimlerChrysler's "U.S. Hybrid Propulsion Systems Development" Subcontract No. ZAN-6-16334-01 with the Midwest Research Institute under DOE Contract No. DE-AC36-83CH10093. Energy management is key to efficient vehicle operation, whether it be between individual battery cells, between the heat engine and the electric drive system or between the total vehicle and the powertrain. DaimlerChrysler's PNGV Concept Vehicle energy management strategy operates at all three levels to maximize overall vehicle efficiency in targeting the PNGV goals.

Starter/Alternator Design for Optimized Hybrid Fuel Economy

2000-11-01 **Technical Paper**

Allan Gale, David Brigham **2000-01-C061**

A Starter/Alternator (S/A) has been developed at Ford Research laboratories for hybrid electric vehicle applications. During development, the vehicle concept of operation and the system performance requirements were used to select the proper technology. The specification development, component selection and subsystem operation process is described. Subsystem performance and vehicle fuel economy are compared and evaluated using hybrid vehicle simulation analysis. These results can be used to identify potential subsystem modifications and alternative vehicle control strategies.

Maximum Electrical Energy Availability With Reasonable Components

2000-11-01 **Technical Paper**

Hélène Cailleux, Bertrand Largy, Charles Vink **2000-01-C071**

The electric power required in automotive systems is quickly reaching a level that significantly impacts costs and fuel consumption. This drives the need to reconsider an electric energy management function. Fast evolving factors such as increasing power usage, and stricter engine management and reliability requirements necessitate a global vehicle approach to energy management. Innovations such as new powernet concepts (42 volt or dual voltage systems), new component technologies (high-performance energy storage, high efficiency and controllable generators), and global electronic and software architecture concepts will enable this new energy management concept. This paper describes key issues to maximize energy availability with reasonable components.

Solid Oxide Fuel Cell Auxiliary Power Unit - A Paradigm Shift in Electric Supply for Transportation

2000-11-01 **Technical Paper**

James Zizelman, Jean Botti, Joachim Tachtler, Wolfgang Strobl **2000-01-C070**

Delphi Automotive Systems and BMW have been jointly developing Solid Oxide Fuel Cell (SOFC) technology for application in the transportation industry primarily as an on-board Auxiliary Power Unit (APU). In the first application of this joint program, the APU will be used to power an electric air conditioning system without the need for operating the vehicle engine. The SOFC-based APU technology has the potential to provide a paradigm shift in the supply of electric power for passenger cars. Furthermore, supplementing the conventional fuel with reformate in the internal combustion engine, extremely low emissions and high system efficiencies are possible. This is consistent with the increasing power demands in automobiles in the new era of more comfort and safety along with environmental friendliness.

Architectural Leadership in the Automotive Industry

2000-11-01 **Technical Paper**

Karl-Thomas Neumann, Hermann Kopetz, Pierre Malaterre, Will Specks **2000-01-C067**

In the new century the automotive industry is transforming itself from an entirely mechanical industry to an industry that is driven by electronics and services. The companies who will be most successful are those who are able to control, drive and renew the architectural concepts enabling the introduction of state-of-the-art information technology to the car and its supporting infrastructure. This paper will first define the term architecture and will elaborate about the increasing relevance of architectural thinking in the automotive domain. Architectural leadership will be defined to mean control (proprietary ownership of components and/or interfaces), creation of a de-facto or legal standard as well as renewal (creation of new products and markets utilizing new linkages of existing architectures). In the second part examples of successful and less successful approaches for establishing architectural leadership in the automotive industry are discussed.

Are Conversations With Your Car Distracting? Understanding the Promises and Pitfalls of Speech-Based Interfaces

2000-11-01 Technical Paper

John D. Lee, Brent Caven, Steven Haake, Timothy L. Brown 2000-01-C012

As computer applications for cars emerge, speech-based interfaces provide an obvious alternative to the visually demanding graphical user interfaces common on desktop applications. However, speech-based interfaces may pose cognitive demands that could undermine driving safety. This study uses a car-following task to evaluate how a speech-based e-mail system affects drivers' response to a periodically braking lead vehicle. A baseline condition with no e-mail system was compared to a simple and a complex e-mail system in both simple and complex driving environments. The results show a 30% (310 msec) increase in reaction time when the speech-based system is present. These results suggest several design strategies to mitigate the distraction potential of speech-based systems.

Car Driver Inactivations in Real-World Precrash Phase

2000-11-01 Technical Paper

Christian Thomas, Jean Yves Le Coz, Yves Page, Anne Damville, Mohamed Kassaagi 2000-01-C007

This study discusses the potential influence of non-driving tasks on the performance of drivers and on the increased risk of involvement in a traffic accident. It is based upon a review of the literature and results of two research projects carried out in France. As a complement to experiments on avoidance maneuvers in a simulated accident situation, subjects were asked to rate both their frequencies and subjective risk level for 18 actions involving secondary tasks such as using a phone when driving. Answers given by French subjects are compared to those given by Japanese subjects. It was clear that actions considered as risky are seldom declared and that secondary tasks are often considered as risky whenever they require hand or visual distraction. The accident sample contains several hundred personal injury car crashes, studied in-depth and on the scene from 1995 to 1999 by a team of accidentologists including a psychologist.

Crashes Induced By Driver Information Systems and What Can Be Done to Reduce Them

2000-11-01 Technical Paper

Paul Green 2000-01-C008

Future in-vehicle information systems may overload drivers, compromising driving safety and product usability. Suggestions of overload appear in (1) statistics from Japan, the United States, and Kuwait for mobile phone-related crashes, (2) statistics from Japan for navigation system-related crashes, and (3) human performance data. From most to least frequent, tasks associated with crashes were receiving a call, dialing, talking (on a phone), looking at a (navigation) display and operating an interface (for navigation). To optimize driver performance for future interfaces, developers should comply with design guidelines (JAMA, SAE J2364), work more closely with human factors experts, expand usability testing, and implement workload managers.

Smart Sensors for Future Robust Systems

2000-11-01 Technical Paper

Michel F. Sultan, David S. Eddy, Brian T. Murray 2000-01-C055

"Smart'' sensor concepts must be considered as the demands of advanced automotive systems increase. These concepts are strongly influenced by the architectural and dependability aspects of future systems. Key features of smart sensors are: communication (two way) with a digital data bus, self- calibration, error source compensation, self-diagnostics, and programmability for "plug and play.'' This paper contains a discussion of the basic future sensor requirements, and it assesses four major sensor technologies with respect to their suitability to meet these requirements. For each technology, the merits and demerits will be reviewed and an example sensing application will be given in order to demonstrate how the technology can be adapted to meet the future requirements.

Cadence Virtual Component Co-Design: An Environment for System Design and Its Application to Functional Specification and Architectural Selection for Automotive Systems

2000-11-01 Technical Paper

Frank Schirrmeister 2000-01-C051

System design is moving from a trial-and-error manual process towards a more rigorous and tool-supported approach. The forcing function for this move is the need to develop new products quickly, correctly and inexpensively. Time-to- market of four to six months are not uncommon today for some electronic systems. At the same time there is a need to leverage maximally what Deep Sub-Micron (DSM) technology has to offer to compete. Hence an approach that has design re- use as its main goal has great potential to increase productivity of the industry as a whole. The basic tenets of the design methodology incorporated into the Cadence Virtual Component Co-Design (VCC) Environment are function- architecture co-design, mapping of functions to flexible and programmable architectures, and efficient and automatic implementation of actual code on the programmable components of the architecture.

Search

Search Results

CRITERIA

Sort by: Relevance Published Title

Text: "international congress on transportation electronics"

Date: 2000

RESULTS

Display: Grid List

Viewing 61 to 73 of 73

Wireless Infrastructure: the Road Map to High Data Rates

2000-11-01 Technical Paper

Mats Nilsson, Lars Nilsson **2000-01-C022**

Second-generation radio access has been a major success story for the global telecommunications industry, delivering telephony and low bit-rate data services to mobile end-users. The growth rate of second-generation mobile telephony indicates that mobile communication is well on its way toward full mass-market penetration. Thanks to the tremendous growth of the Internet, multimedia is also penetrating the mass market at an explosive pace. Combined, the digital cellular footprint and the multimedia services of the Internet form the basis of tomorrows integrated wireless. The transition to third-generation capabilities must be based on a feasible migration path that defines a way of integrating multimedia, packet switching and wideband radio access into the dominating second-generation systems of our day.

Suppression Technologies for Advanced Air Bags

2000-11-01 Technical Paper

Santosh Anishetty, Morgan Murphy, Craig Tieman, Chris Caruso **2000-01-C037**

In May 2000 the National Highway Traffic Safety Administration (NHTSA) issued the final rule for the Advanced Air Bag regulations effective MY 2004 for vehicles to be sold in the United States. These regulations are in response to the air bag-induced injuries seen in the field, especially to children and short women. Advanced air bags require a vehicle manufacturer to design air bags for a broad array of occupants: 12-month-old, 3-year-old and 6-year-old children, and 5th percentile adult females, as well as 50th percentile adult males with new and more stringent injury criteria. Requirements for minimizing air bag risks include automatically turning off the air bag in the presence of young children or deploying the air bag in a manner much less likely to cause serious or fatal injury to out-of-position occupants. Technologies that disable the air bag in the presence of young children or adults in out-of-position are termed as "suppression technologies.'

Optimizing Distributed Systems for Automotive E/E Architectures

2000-11-01 Technical Paper

Peter Abowd, Gary Rushton, Viren Merchant **2000-01-C083**

The rapid growth of vehicle feature content continues to challenge automotive designers. The total vehicle feature content seriously impacts the manufacturing complexity of any single vehicle. Traditional strategies for introducing new features into high-content luxury vehicles before moving the feature into economy vehicles have been undermined by the fast moving consumer electronics field. The challenge for automotive OEM and Tier 1 suppliers is to optimize the vehicle architecture in order to provide more efficient means of introducing features expediently and efficiently. Therefore, any production vehicle's Electrical, Electronic, & Software (EES) architecture must successfully support modular sourcing, modular assembly, global manufacturing schemes, cost and weight issues.

A New Design for Automotive Alternators

2000-11-01 Technical Paper

David J. Perreault, Vahe Caliskan **2000-01-C084**

This paper introduces a new design for alternator systems that provides dramatic increases in peak and average power output from a conventional Lundell alternator, along with substantial improvements in efficiency. Experimental results demonstrate these capability improvements. Additional performance and functionality improvements of particular value for high-voltage (e.g., 42 V) alternators are also demonstrated. Tight load-dump transient suppression can be achieved using this new design and the alternator system can be used to implement jump charging (the charging of the high-voltage system battery from a low-voltage source). Dual-output extensions of the technique (e.g., 42/14 V) are also introduced. The new technology preserves the simplicity and low cost of conventional alternator designs, and can be implemented within the existing manufacturing infrastructure.

Fast and Reliable Process for the Development of Automotive Embedded Software

2000-11-01 Technical Paper

S. Boutin, J.-F. Tarabbia **2000-01-C080**

The optimization of the electronics architecture of the vehicle requests flexibility in introducing new functionalities at a competitive cost. Within a consortium project called "Embedded Electronics Architecture," two car manufacturers: Renault and PSA, Aerospatiale, three tier 2 suppliers: Sagem, Siemens and Valeo, three R&D labs: INRIA, IRCyN and Loria have joined their R&D efforts. The partners define a fast and reliable process for designing the system architecture, developing the associated embedded software, selecting the appropriate hardware and software allocation, defining and integrating reusable components. The development process is based on independence between hardware and software, standard components, development methods and tools, contactualization of the development issues between partners.

Low-Power Flexible Controls Architecture for General Motors Partnership for a New Generation (Pngv) Precept Vehicle

2000-11-01 Technical Paper

Joe LoGrasso, Kevin Kidston, Walton Fehr **2000-01-C060**

The complexity of designing and implementing a vehicle electrical control system for ultra fuel-efficient hybrid vehicles is significantly greater than that of a conventional vehicle. To quickly demonstrate and iterate capabilities of these vehicles, an efficient and rapid means for developing requirements, mapping these into an electrical control and communications architecture, and developing prototype systems is needed. The General Motors Precept concept vehicle is an example of an energy- efficient vehicular control system developed using a "requirements to software'' development process and electronic controller infrastructure that demonstrates these attributes. The Precept is General Motors Corporation's technology demonstration concept vehicle developed to address General Motors Corporation's commitment to the Partnership for a New Generation (PNGV) program.

Driver Performance Improvement Through the Driver Advocate: a Research Initiative Toward Automotive Safety

2000-11-01 Technical Paper

Don Remboski, Judy Gardner, David Wheatley, Joshua Hurwitz, Tom MacTavish, Robert "" Mike'' Gardner **2000-01-C075**

The flood of information reaching drivers - telematics, infotainment, collision warning and others - requires a new look at the car's user interface. Using concepts from affective computing, social sciences, cognitive psychology, and more traditional human-machine interface design, the Driver Advocate, a driver assistance interface being developed by Motorola, orchestrates the driver's attention to the most important task in a useful and acceptable manner. The Driver Advocate monitors all driver-related controls along with information from telematics, navigation and chassis control systems. For example, during a high workload situation the Driver Advocate will withhold

non-essential incoming information until the driver has reacted to mission critical information, thus minimizing potential driver distraction. The purpose of this paper is to outline the research Motorola has initiated to better define the emerging opportunities and challenges.

The Significance of Navigation Map Databases in Advanced Driver Assistance Systems and Dynamic Route Guidance

2000-11-01 **Technical Paper**

M. Salahuddin Khan, Michele Roser Herbst **2000-01-C076**

Navigation quality map databases have evolved during the last 10 years in coverage, content and accuracy to the level today that they are being used (and considered) for applications beyond turn-by-turn navigation. For a number of advanced driver assistance systems in Europe and North America, the navigation maps are being included as an essential component for efficient and economical functionality. In addition, for dynamic route guidance, referencing incident location, extent of delay, traffic speed and on-the-move re-routing require the most sophistication in road network data and routing algorithms.

Identification of the Optimum Vehicle Class for the Application of 42v Integrated Starter Generator

2000-11-01 **Technical Paper**

John M. Miller, Ken Hampton, Robert Eriksson **2000-01-C073**

Today nearly all automotive manufacturers are developing motor-generator systems for improved fuel economy by implementing idling-stop and other power train enhancements. It is said that powertrain technology has always pioneered the development of automotive electronic control throughout history. The integrated starter generator (ISG) promises to expand the scope of powertrain control further through fuel economy improvement, emissions reduction, longitudinal vehicle dynamics improvement and customer feature enhancements. At the present time the cost imposed by usage of an ISG system is very high due mainly to its need for a power optimized 42V battery and high power electronics. This paper takes a critical look at the vehicle benefits attributable to ISG and its implementation costs over various vehicle classes.

Powertrain System Design: Functional and Architectural Specifications

2000-11-01 **Technical Paper**

G. Bombarda, G. Gaviani, P. Marceca **2000-01-C049**

Powertrain controller design is among the most challenging problems due to the complexity of the functions that the system has to support, to the safety aspects and to the cost limits imposed by car manufacturers. To compound these difficulties, time-to-market requirements are becoming more and more stringent. Design time, continuously changing specifications and safety considerations have pushed the design more and more towards software implementation of the main functionality. Software has been traditionally designed with very little abstraction in mind thus forcing a tight dependency of the implementation on the particular hardware architecture, e.g., the instruction set of the micro- controller. Software legacy has made the rapid adoption of new technology and IC's almost impossible, stifling innovation. In addition, the absence of a correct abstraction hierarchy made verifying the correctness of the behavior of the system as well as adding new functionality extremely difficult.

Future Automotive Multimedia Subsystem Interconnect Technologies

2000-11-01 **Technical Paper**

Akram M. Mufid **2000-01-C028**

For the past decade or so, automotive entertainment subsystem architectures have consisted of a simple Human Machine Interface (HMI), AM-FM tuner, a tape deck, an amplifier and a set of speakers. Over time, as customer demand for more entertainment features increased, automotive entertainment integrators made room for new features by allowing for the vertical integration of analog audio and adding a digital control. The new digital control came to entertainment subsystems via a low-speed multiplexing scheme embedded into the entertainment subsystem components, allowing remote control of these new features. New features were typically incorporated into the entertainment subsystem by independently packaging functional modules. Examples of these modules are cellular telephone, Compact Disc Jockey (CDJ), rear-seat entertainment, Satellite Digital Audio Radio System (S-DARS) receiver, voice and navigation with its associated display and hardware.

Resource Management for Third-Generation Telematics Systems

2000-11-01 **Technical Paper**

Robert Allen Gee, Stan Gonsalves **2000-01-C021**

Due to the safety-related nature of in-vehicle systems and the increasing complexity of distributed telematics software services, protocols, and communications mechanisms, the need exists for a comprehensive, extensible approach to manage and prioritize applications and resources. The characteristics for such an approach include predictability, support for the user experience, error handling and preemptive correction, and support for unknown future applications and capabilities. This paper provides a structure for addressing telematics services, discusses the characteristics required for application and communications resource prioritization and management, and describes an extensible approach for managing complex interactions in current and future telematics systems.

Automotive Electronics: Trends and Challenges

2000-11-01 **Technical Paper**

Alberto Sangiovanni-Vincentelli **2000-01-C047**

The car as a self-contained microcosm is undergoing radical changes due to the advances of electronic technology. We need to rethink what a "car" really is and the role of electronics in it. Electronics is now essential to control the movements of a car, of the chemical and electrical processes taking place in it, to entertain the passengers, to establish connectivity with the rest of the world, to ensure safety. What will an automobile manufacturer's core competence become in the next few years? Will electronics be the essential element in car manufacturing and design? We will address some of these issues and we will present some important developments in the area of system design that can strongly impact the way in which a car is designed.