國立交通大學

電控工程研究所

碩 士 論 文

近代汽車 **MOST** 與 **CAN** 網路介面之整合
Integration of MOST & CAN Networks in Modern Automobiles

研 究 生：陳奕祥

指導教授：王啟旭　教授

中華民國九十八年九月

Integration of MOST & CAN Networks in Modern Automobiles

研 究 生：陳奕祥　　Student：I-Hsiang Chen

指導教授：王啟旭 教授　　Advisor：Chi-Hsu Wang

國 立 交 通 大 學

電控工程研究所

碩 士 論 文

A Thesis

Submitted to Institute of Electrical and Control Engineering

College of Electrical Engineering and Computer Science

National Chiao-Tung University

In Partial Fulfillment of the Requirements

for the Degree of Master

in

Electrical and Control Engineering

September 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年九月

近代汽車 MOST 與 CAN 網路介面之整合

研究生：陳奕祥　　　　　指導教授：王啟旭 教授

國立交通大學電控工程研究所

國立交通大學

摘　要

本論文主要在探討一種為汽車產業開發的多媒體網路技術--MOST (Media Oriented System Transport，多媒體導向系統傳輸)，和汽車網路 CAN(Controller Area Network)。我們將示範其多媒體應用及資料傳輸控制介面的應用，共利用 OASIS SiliconSystems 公司所提供的 MOST 網路服務應用程式介面 (MOST NetServices API)，來撰寫程式去控制在 MOST 網路上的多媒體設備，例如 DVD 播放器、收音機、IPod Gateway 及聲音放大器。在應用中，我們將個人電腦主機當作網路的中控台，透過 MOST 網路介面卡 (MOST PCI Board)或光纖網路分析介面盒 (Optolyzer Interface Box)及 USBCAN2 分析工具，來建立 MOST 網路與 CAN 網路和個人電腦主機之間的通訊。藉由各種多媒體應用之介紹及 SAAB 95 SID 的應用，說明 MOST 網路服務應用程式介面的使用方式及 USBCAN2 的使用方法；在最後，我們設計了一個可同時控制 MOST 與 CAN 網路的程式介面，來實現在 MOST 網路應用下的影音伺服器及 CAN 網路下的設備操作。

Integration of MOST & CAN Networks in Modern Automobiles

Student：I-Hsiang Chen　　　　Advisor：Chi-Hsu Wang

Institute of Electrical and Control Engineering

National Chiao Tung University

ABSTRACT

The main purpose of this thesis is to explore and demonstrate CAN(Controller Area Network) and the multimedia applications under MOST (Media Oriented System Transport) network which is originally developed for the automotive industry. With the MOST NetServices API provided by OASIS SiliconSystems, we can build programs to control multimedia devices for MOST, such the DVD Player, Radio Tuner, IPod Gateway and Audio Amplifier. Besides, both the MOST PCI Board and the Optolyzer Interface Box and USBCAN2 are used for establishing the communication between the MOST network, CAN network and a PC which serves as a control central for the network. The details of understanding the NetServices API under MOST network is explained through different multimedia applications. The understanding of USBCAN2 is explained through demonstrating SAAB 95 SID. We show the application of USBCAN2 device by demonstrating SAAB 95 SID. In addition to the ordinary MOST and CAN applications, a new Audio/Video server interface under MOST network is also designed.

誌　　謝

首先感謝我的指導老師　王啟旭教授兩年來細心教導，以及在我的學習過程中給予的支持與鼓勵，除了讓我在專業領域上有更深刻的體會外，在待人處世和學習態度方面也都獲得相當大的啟發，也讓我理解到做研究應有的態度與方法。

另外要特別感謝實驗室學長文榮生、黃得裕、洪堃能、王之政、鍾元之提供研究上的許多寶貴意見與經驗，也感謝實驗室同窗林宗賢、黃泰鈞和實驗室學弟劉丞偉、黃繼輝的陪伴與支持，並在生活與研究上互相幫忙。還要感謝實驗室學長黃俊穎的指導，在學長之前經驗累積的基礎上，我才能在論文研究有如此的進步。

最後感謝家人的支持與關懷。我的父母在我的求學過程中毫無保留的付出，總能給予我精神上最大的鼓勵，讓我能夠在這兩年內無後顧之憂的求學。

陳奕祥　于新竹

中華民國九十八年九月

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

In the master thesis of my senior, Jun-Ying Huang [1], several multimedia applications of MOST (Media Oriented System Transport) are developed with the MOST NetServices API [2] (Application Programming Interface) except IPod Gateway. After his graduation, I took over the experiments of MOST in our lab. Continuing with Jun-Ying Huang's work, and study CAN network in this master thesis, the control program for multimedia applications of MOST will be explored and enhanced, which contributes the design of Audio/Video server under MOST network.

1.1 MOST & CAN history with the objective of this MS thesis

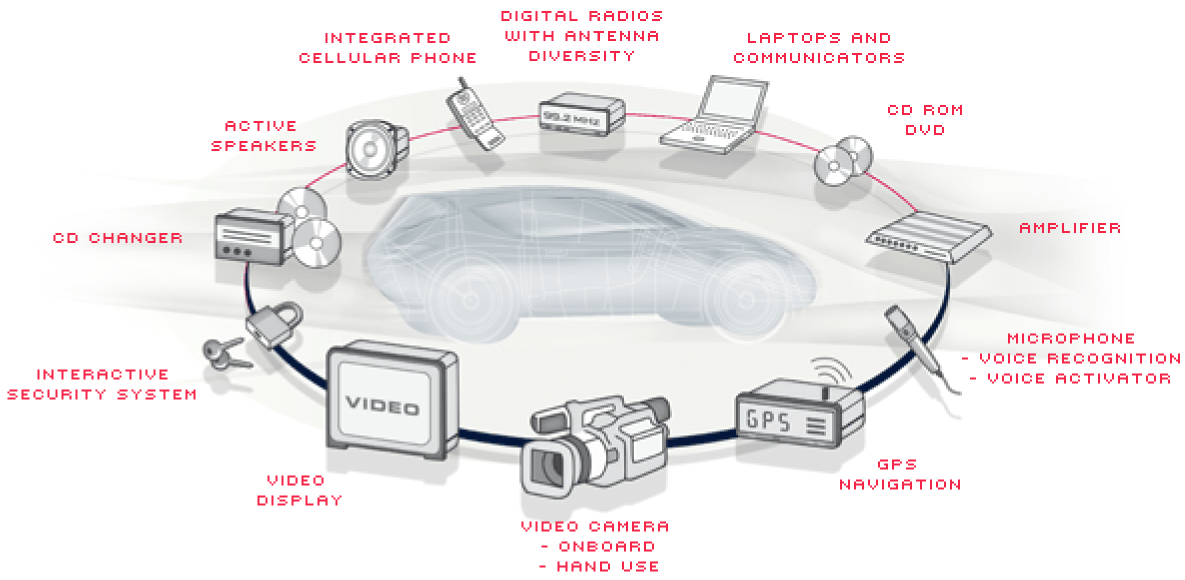

Figure 1.1 Multimedia and Communication peripherals

MOST (Media Oriented Systems Transport) is a multimedia network developed for the automotive industry. In-vehicle consumer devices such as DVD players, MP3 players, IPod players, GPS systems, car phones and Bluetooth devices can be linked together to work as a single system on the MOST network. As show in figure 1.1 [3] [4]. With standardized connectors and media e.g., low cost Plastic Optical Fiber (POF) or Unshielded Twisted Pair (UTP), high bandwidths up to 25 Mbps and 150 Mbps will support in the future.

MOST technology not only includes the physical connection between devices, but also defines properties and methods for devices to interact with each other. The software interfaces allow applications running on different devices to communicate and exchange information. A transport mechanism that sets up a link for streaming data between devices is also defined.

The CAN (Controller Area Network) [5] is a serial communications protocol which supports distributed control with a very high level of security (Multi-master hierarchy, Broadcast communication, Error detecting and re-transmission) [6].

CAN is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other within a vehicle without a host computer. It was designed specifically for automotive applications but is now also used in other areas.
Development of the CAN-bus started originally in 1983 at Robert Bosch GmbH. The protocol was officially released in 1986 at the Society of Automotive Engineers (SAE) [7] congress in Detroit, Michigan. The first CAN controller chips, produced by Intel and Philips, came on the market in 1987. Bosch published the CAN 2.0 specification in 1991.

Feature

- Message-oriented transmission protocol.
- Message contains the priority of the message.
- Easy to add a receiver station to an existing CAN network.

1.2 Scope of work

The MOST devices in our lab, including the RadioTuner4MOST, Amplifier4MOST, DVDPlayer4MOST, IPod Gateway, the MOST PCI Board and the Optolyzer Interface Box, from SMSC, are connected in a ring topology as a multimedia network system. Via the MOST PCI Board, a PC is capable of communicating with the MOST network for management. Moreover, by means of the NetServices API, we can develop our control programs on the Windows platform to implement different multimedia applications of MOST. Compared with Jun-Ying Huang's work, in this thesis, more functions will be integrated in the control program.

The CAN devices in our lab, including USBCAN2 interface, SAAB 95 SID, and NI-CAN PCI board, which are connected through two twisted wires of low speed CAN-bus. Via USBCAN2, a PC is capable of controlling other CAN devices by sending CAN messages.

CHAPTER 2

MOST Specification

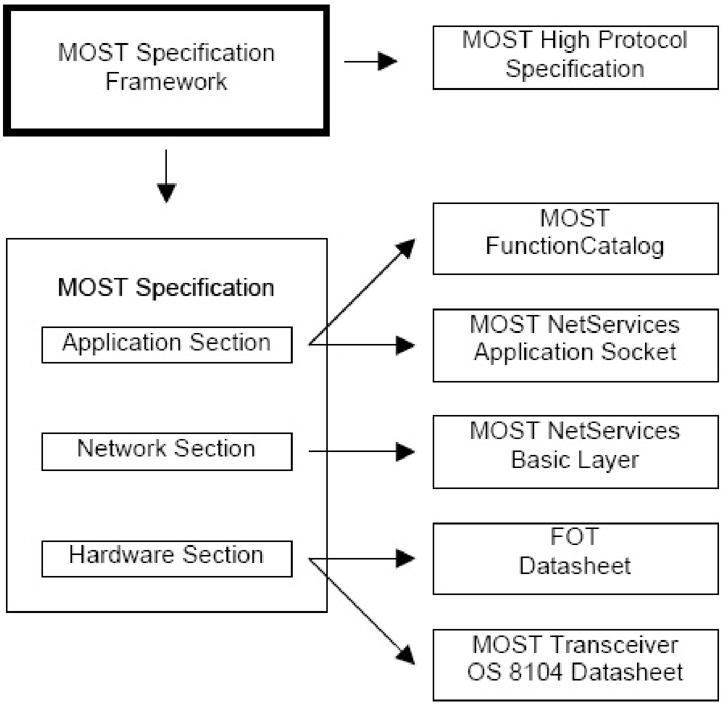

Figure 2.1 Structural overview of the documentation

Figure 2.1 [8] is the structural overview of the MOST specification documentation. It is a main specification within the MOST Framework. This specification is divided into three parts, Application Section, Network Section, Hardware Section. The arrows show the direction of references.

In this chapter, based on the documents "MOST Specification Framework" [9] and "MOST Specification" [10], we wish to give an introductory overview of the MOST System and its abilities. A MOST System often indicates a MOST network which is a group of MOST Devices linked together by some way.

A MOST System can be described by three definition areas which are MOST Interconnect, MOST System Services, and MOST Devices.

At first, we start with MOST Topology, the way of physical interconnection between nodes. Then we get in the section of Data Transport, to see what is transported on the MOST network.

The third part of this chapter is Logical Device Model in which the logical components of a MOST device are described along with their functionalities. This section covers the contents from the application level to the lower physical layer. [11]

2.1 MOST structure

2.1.1 Point to Point Link

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. However, this basic configuration can easily be extended to more complex structures, such as branches or bi-directional links.

2.1.2 Ring Topology

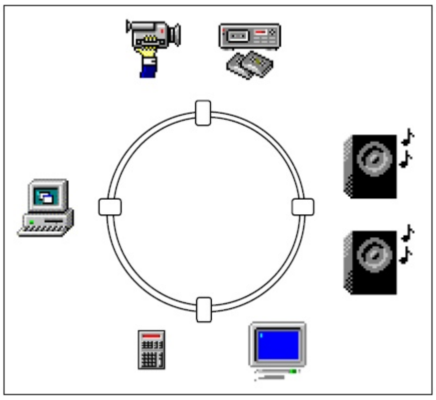

Figure 2.2 Ring Topology

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, as shown in Figure 2.2.

There are many advantages of a ring topology, which include the following:

- Minimum use of physical layer media, reducing system cost and weight.
- 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.
- 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.

One major disadvantage of a traditional ring topology is that the failure of one node can cause the entire network to fail. However, it can be largely overcome in a MOST network. 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.

2.2 MOST protocols

2.2.1 Structure of MOST Protocols

On the application level, the functions are addressed independently of the devices they belong to. Furthermore, the particular functions are unified into the function blocks according to their content. Hence, a function is addressed in a function block. The MOST protocols on the application layer have the following structure: [12]-[14]

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

DeviceID

The DeviceID with a length of 16 bits, stands for a physical device or a group of devices (Group Address), and is network specific. It represents the logical node address of either the sender or the receiver, depending on the case. A group address or a broadcast address (0x03C8) could be used for the target too. In case the sender does not know the receiver's address, the DeviceID is set to 0xFFFF and will be corrected by the NetServices of the sender.

FBlockID

It specifies particular FBlock in the device. Every function block with a special FBlockID must contain certain specific functions other than the mandatory functions. Two kinds of proprietary FBlockIDs are defined: System specific and Supplier specific. System specific type can be used by any carmaker and is predefined, where the Supplier specific type is used by OEMs (Original Equipment Manufacturer) for development purpose.

InstID

It is an identifier of instance of function block. If the FBlockID is not unique within the system, this identifier specifies the function block. FBlockID and InstID create the functional address which has to be unique.

- **0x01 -** By default, every function block has instance ID 0x01. In case there are several FBlocks of the same kind within one MOST device, the default number (within the device) starts at 1.
- **0x00 -** Don't care. The device dispatches the message to one specific FBlock in the device.
- **0xFF -** Broadcast (within a device). The message is dispatched to all instances of the matching FBlock.

FktID

The FktID stands for a function. This means a function unit (Object) within a device which provides operations (e.g. "Eject CD") can be called via the network. The FktID is encoded in 12 bits on network level and is extended to 2 bytes on application level. The address range of FktIDs is subdivided into the following sections: Coordination, Mandatory, Extensions, Unique, System Specific and Supplier Specific.

OPType

The OPType indicates which operation must be applied to the property or method specified in FktID:

Length

Length is encoded in 16 bits which specifies the length of the data field in bytes. This parameter is not transferred through the control channel, but is computed in the receiving node.

Data is transferred in a continuous bi-phase encoded bit stream yielding more than a 24.8Mbps data rate, and the sample frequency in a MOST system can be chosen in a range between 30 kHz and 50 kHz.

2.2.2 Data Types

Different types of information can be transmitted over MOST and the frame structure separates the data into three different sections: Control data, asynchronous packet data and synchronous stream data.

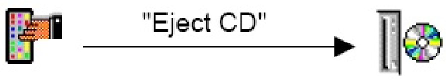

Figure 2.3 Control data

- **Control data transfer** is for device specific transfers and system management. As shown in Figure 2.3, control data "Eject CD" is sent to the CD player to eject the CD. This kind of data is transported in parallel to the real-time and asynchronous data.

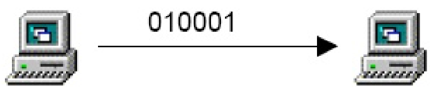

Figure 2.4 Packet data

- **Bulk data / burst data transfer** in asynchronous packets with variable bandwidth requirements. As shown in Figure 2.4, it is often used for TCP/IP packets between computers.

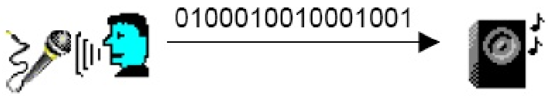

Figure 2.5 Synchronous data

- **Real-time data transfer** which requires a guaranteed bandwidth to maintain data quality. As shown in Figure 2.5, audio stream from microphone to the speaker is such kind of data. Indeed, all nodes on the MOST network have access to it.

2.2.3 Frame Structure

A block consists of 16 frames with 512 bits (64 bytes) each, running at the system sample rate as frame rate (typical 44.1 kHz). Organization of data transfer in blocks of frames is required for network management and control data transport tasks. A control message has a fixed size of 32 bytes and is time multiplexed over 16 frames (1 block), each having 2 control data bytes.

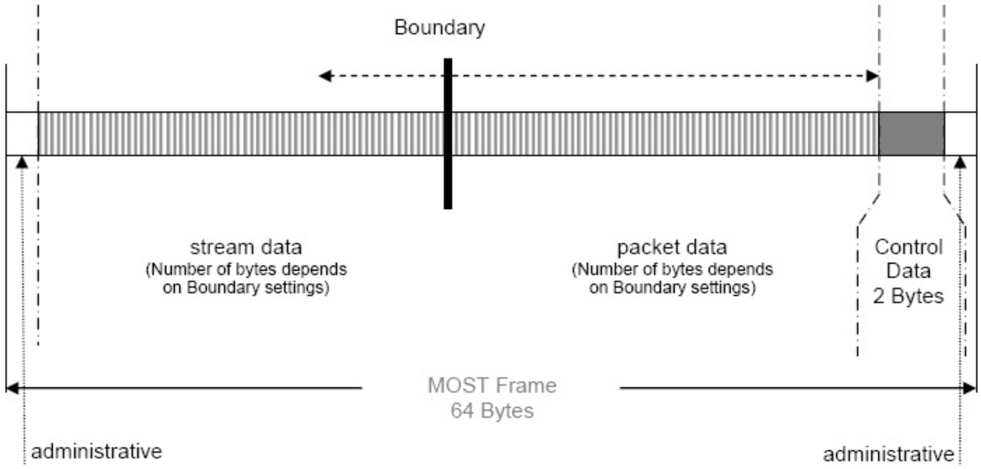

Figure 2.6 MOST Frame

Byte Number	Task
0	Administrative ◦ Preamble (bits 0-3) ◦ 4 bits Boundary Descriptor (bits 4-7)
1 - 60	60 data bytes
61 - 62	2 data bytes for Control Messages
63	Administrative ◦ Frame control and status bits ◦ Parity bit (last bit)

Figure 2.7 Structure of MOST Frame

One MOST25 frame consists of 512 bits (64 bytes). The first byte is used for administrative purposes. The next 60 bytes are used for stream and packet data transfer, where the Boundary is defined in 4 byte steps. All Stream data bytes are transmitted before any Packet data bytes occur. The next two bytes of each frame are reserved for Control data and the last byte is another administrative byte. As shown in Figure 2.6 and Figure 2.7.

2.3 MOST NetServices

This section consists of two parts. Section 2.3.1 gives an overview of MOST System Services in which MOST NetServices are included. It can be regarded as an introduction to the MOST Specification from a different point of view. In Section 2.3.2, MOST NetServices Layer 1 API is introduced, since we will construct our multimedia control program with the API. We survey MOST System Services and Basic Layer System Services API in this chapter, and would not go deep into the specifications.

2.3.1 System Services Overview

The MOST System Services provide all basic functionality to operate a MOST System, and are consists of the following four parts:

- **Application Socket**

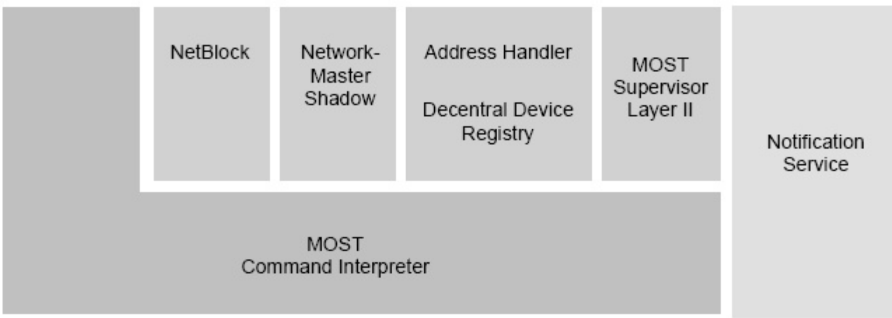

Figure 2.8 Application Socket

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 (Layer 2) together with the Basic Layer System Services

(Layer 1) are implemented in the NetServices software which contains all basic functionality a MOST Device needs.

- **Basic Layer System Services**

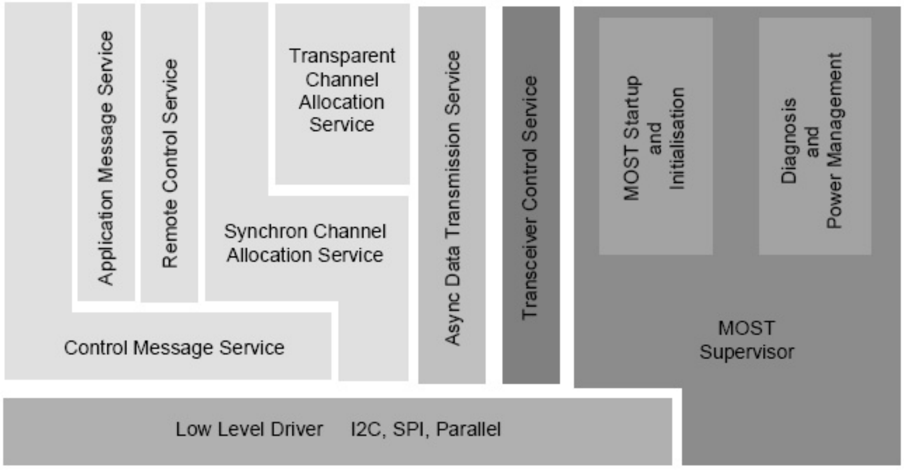

Figure 2.9 Basic Layer System Services

The Basic Layer of the System Services is also known as MOST NetServices Layer 1 which provides comfortable access on the Low Level System Services.

- **Low Level System Services**

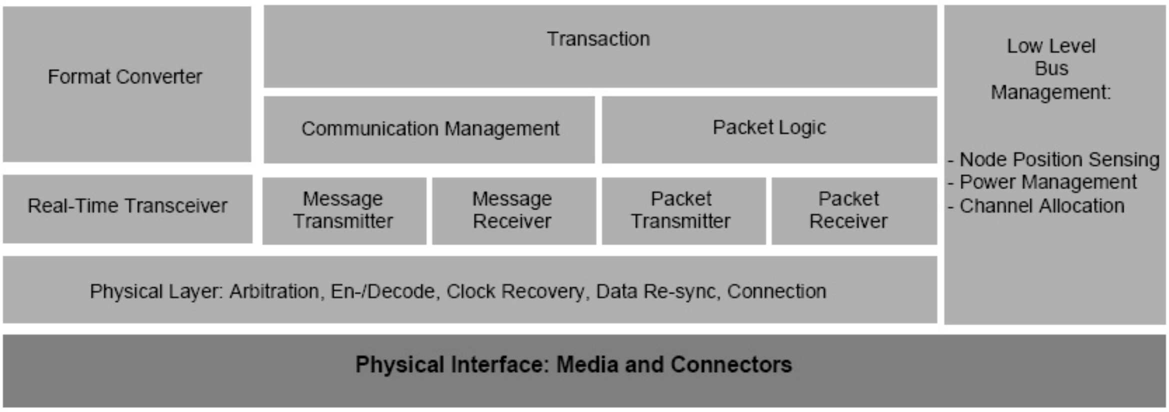

Figure 2.10 Low Level System Services

Low Level System Services, except for the physical interface, are implemented in the MOST Transceiver [8].

- **Stream Services**

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

2.3.2 NetServices Layer 1 API

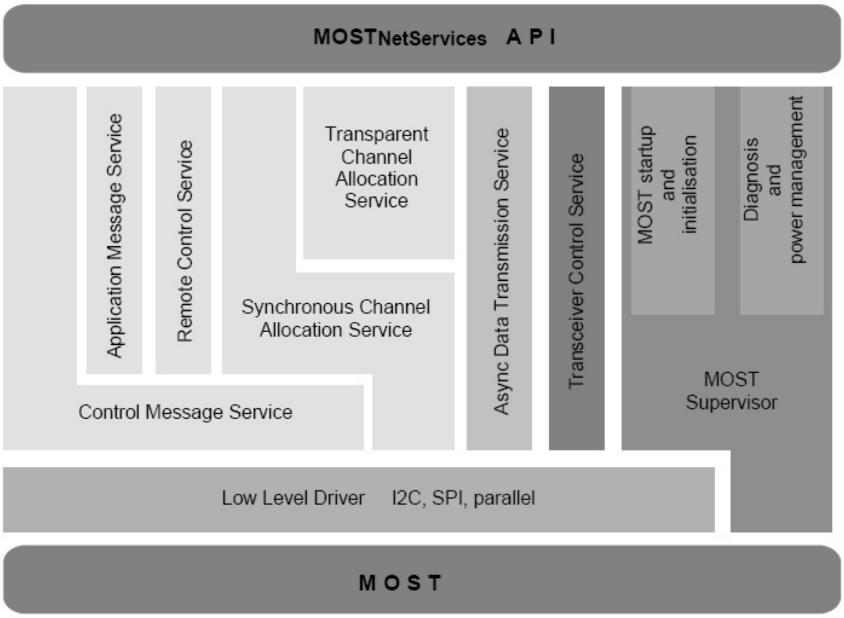

Figure 2.11 Structure of MOST NetServices Layer 1

MOST NetServices Layer 1 API is the interface between the MOST Low Level System Services and an application. All functions are combined in a library that they can be included if required. MOST NetServices API is implemented in ANSI C and can be adapted by the file "*adjust.h*" which contains all the parameters needed to set up NetServices according to the application.

- MOST NetServices Kernel (MNS)
  - Initialization of all services
  - NetServices trigger functions

- MOST Supervisory and Startup Functions (MSV)
  - Transceiver Initialization

- Master/Slave Selection
- Compiler Selectable Source Port Configuration
- Network Start Up and Shut Down
- Power Management
- Failure Diagnosis and Reporting
- Self Testing Services

- Control Message Service (CMS)
  - Initialization
  - Control Message Receive Service
  - Control Message Transmit Service
  - Message Buffering and Handling
  - Error Handling and Notification

- Application Message Service (AMS)
  - Initialization
  - Application Message Send Handler
  - Application Message Receive Handler
  - Message Buffering and Handling
  - Error Handling and Notification

- Remote Control Service (RCS)
  - Initialization
  - Remote Write Service
  - Remote Read Service
  - Remote Error Handling and Notification

- Synchronous Channel Allocation Service (SCS)

- Channel Allocation
- Allocation and Routing
- De-Allocation
- De-Allocation and Routing Disconnect
- Source Connect
- Source Disconnect
- Sink Connect
- Sink Disconnect
- Detect Channel by Label

- Transparent Channel Allocation Service (TCS)
  - Channel Allocation
  - Allocation and Routing
  - De-Allocation
  - De-Allocation and Routing Disconnect
  - Source Connect
  - Source Disconnect
  - Sink Connect
  - Sink Disconnect

- Asynchronous Data Transmission Service (ADS)
  - Initialize – SetDataAddress
  - Buffered Transfer of Data Packages
  - Buffered Receive of Data Packages
  - Data Transfer Error Handling

- MOST Transceiver Control Service (MCS)
  - Addressing

- Accessing important Transceiver's Register
- Selecting RMCK Frequency
- Master Functions

2.4 MOST Multimedia Applications

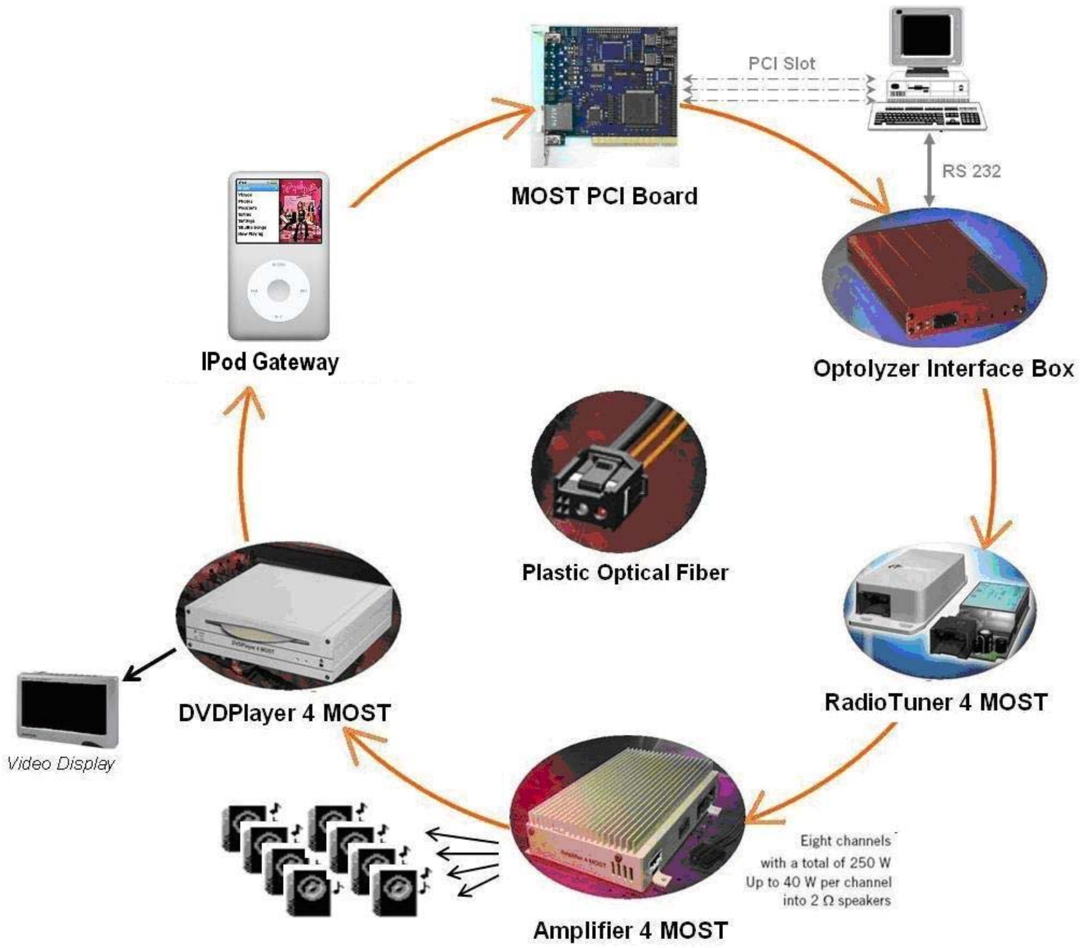

Figure 2.12 Structure of MOST Network

2.4.1 Application 1: Listen to the Radio

Source: RadioTuner4MOST

Sink: Amplifier4MOST

- Listen to the radio on the MOST network.

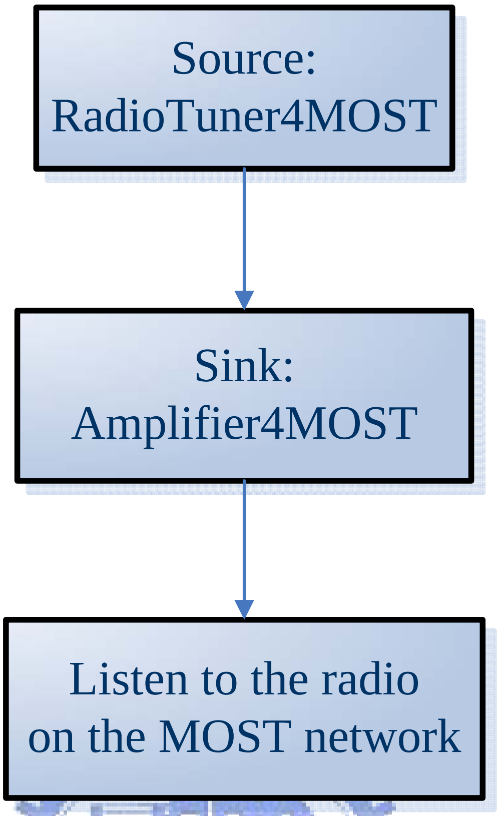

Figure 2.13 Flowchart of listening to the Radio

2.4.2 Application 2: Watch DVD movies

Source: DVDPlayer4MOST

Sink: Amplifier4MOST

- Watch VCD/DVD movies.

  (Video is output from the DVD player while audio is output from the amplifier.)

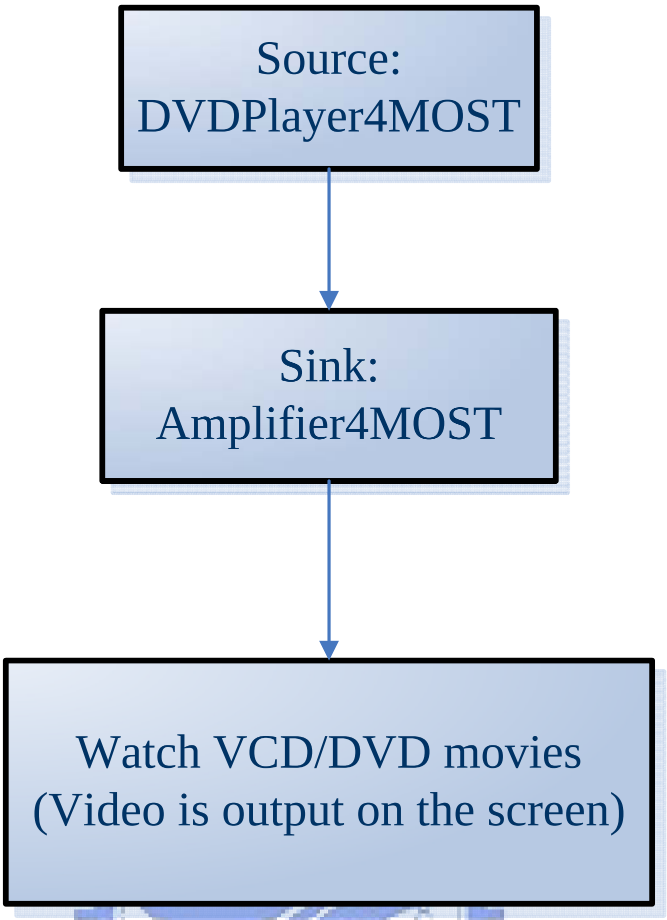

Figure 2.14 Flowchart of watching DVD movies

2.4.3 Application 3: Listen to IPod music

Source: IPod Gateway

Sink: Amplifier4MOST

- Listen to IPod on the MOST network.

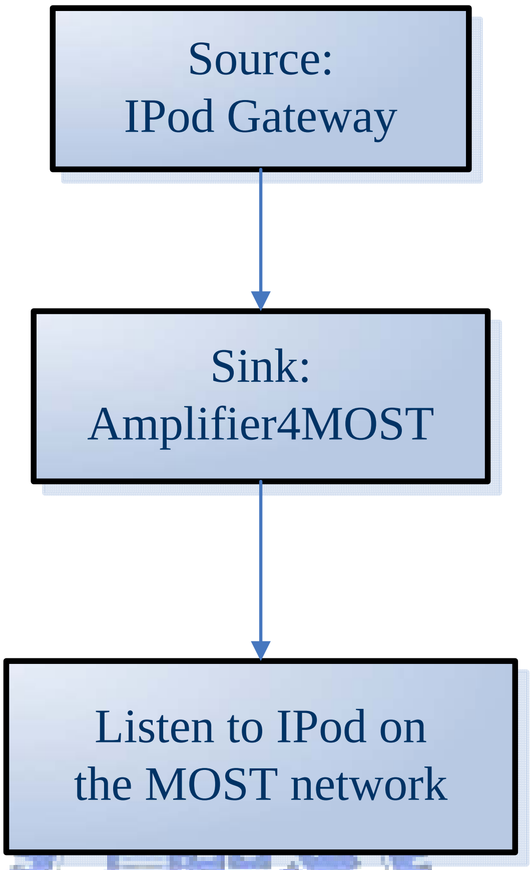

Figure 2.15 Flowchart of listening to IPod music

CHAPTER 3

CAN specification

CAN (Controller Area Network) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other within a vehicle. It was designed specifically for automotive applications but is now also used in other areas.

3.1 CAN structure

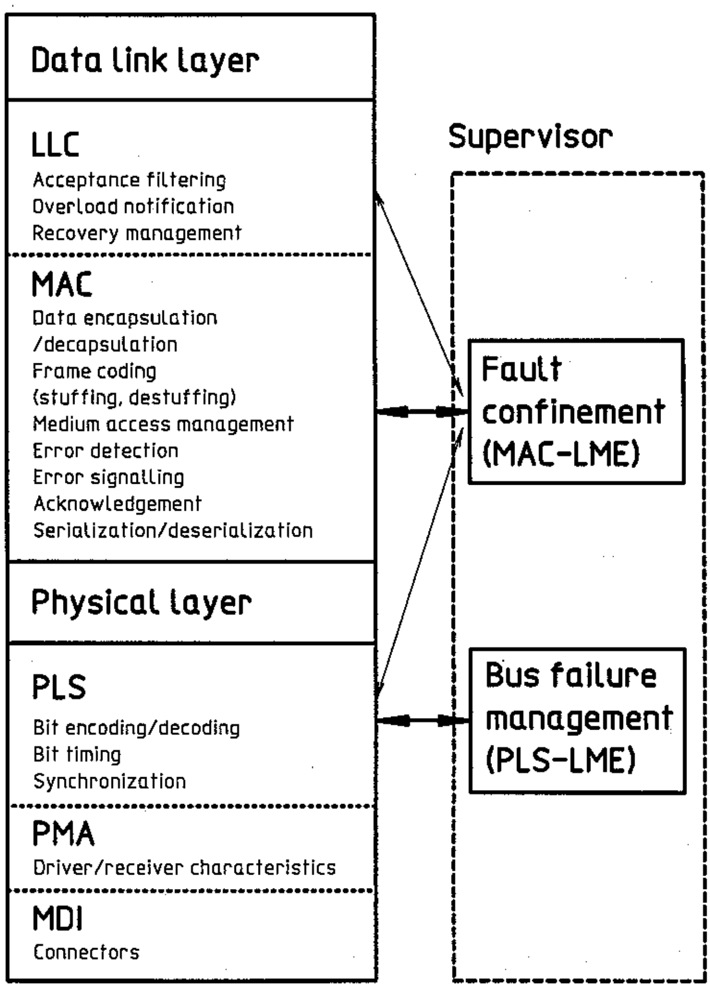

Figure 3.1 Layer architecture of CAN

The CAN architecture represents two layers: [15]

- Data link layer

- LLC (Logic Link Control)
- MAC (Medium Access Control)

- Physical layer
  - PLS (Physical Signalling)
  - PMA (Physical Medium Attachment)
  - MDI (Medium Dependent Interface)

3.1.1 Bus topology

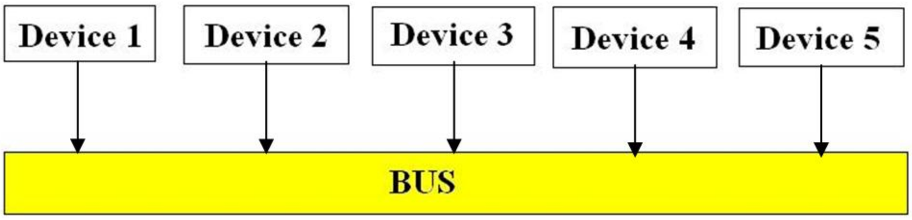

Figure 3.2 Bus topology

All CAN devices connect to the same transmission line by using two twisted wires. As shown in Figure 3.2.

3.2 CAN protocols

3.2.1 Structure of CAN protocols

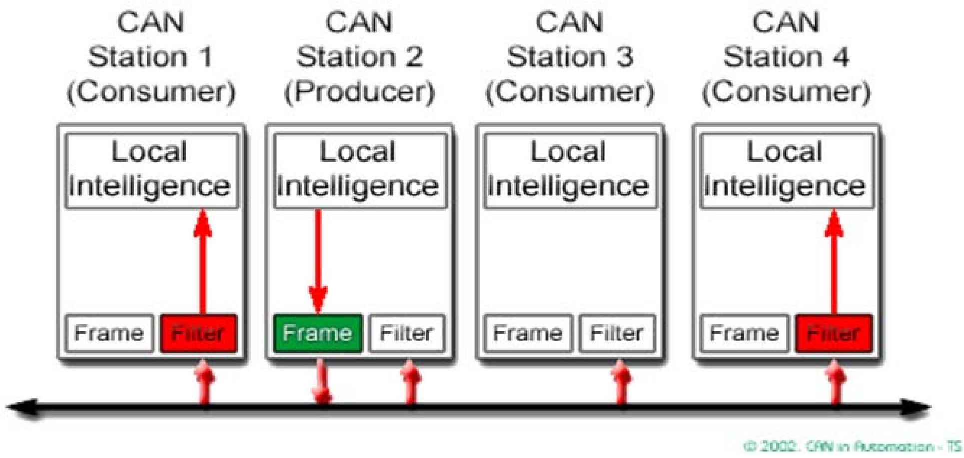

Figure 3.3 Principles of CAN data exchange

Each node requires: [5]

a host-processor

The host-processor decides what received messages mean, and which messages it wants to transmit itself

Sensors, actuators and control devices can be connected to the host-processor (if desired).

a CAN Controller (hardware with a synchronous clock)

Receiving: the CAN Controller stores received bits (one by one) from the bus until an entire message is available, that can then be fetched by the host processor (usually after the CAN Controller has triggered an interrupt)

Sending: the host-processor stores its transmit-messages into a CAN Controller, which transmits the bits serially onto the bus

a Transceiver (possibly integrated into the CAN Controller)

Receiving: it adapts signal levels from the bus, to levels that the CAN Controller expects and has protective circuitry that protect the CAN Controller

Sending: it converts the transmit-bit signal received from the CAN Controller into a signal that is sent onto the bus

Bit rates up to 1 Mbit/s are possible at network lengths below 40 m. Decreasing the bit rate allows longer network distances (e.g. 125 kbit/s at 500 m).

3.2.2 CAN Technology

CAN is a multi-master broadcast serial bus standard for connecting electronic control units (ECUs).

Each node is able to send and receive messages, but not simultaneously: a message (consisting primarily of an ID — usually chosen to identify the message-type/sender — and up to eight message bytes) is transmitted serially onto the bus, one bit after another — this signal-pattern codes the message (in NRZ) and is sensed by all nodes.

The devices that are connected by a CAN network are typically sensors, actuators and control devices. A CAN message never reaches these devices directly, but instead a host-processor and a CAN Controller is needed between these devices and the bus.

If the bus is free, any node may begin to transmit. If two or more nodes begin sending messages at the same time, the message with the more dominant ID (which has more dominant bits i.e. bit 0) will overwrite other nodes less dominant IDs, so that eventually (after this arbitration on the ID) only the dominant message remains and is received by all nodes.

3.2.3 Data Types

A CAN network can be configured to work with two different message (or "frame") formats: the standard or base frame format (or CAN 2.0 A), and the extended frame format (or CAN 2.0 B). The only difference between the two formats is that the "CAN base frame" supports a length of 11 bits for the identifier, and the "CAN extended frame" supports a length of 29 bits for the identifier, made up of the 11-bit identifier ("base identifier") and an 18-bit extension ("identifier extension"). The distinction between CAN base frame format and CAN extended frame format is made by using the IDE bit, which is transmitted as dominant in case of an 11-bit frame, and transmitted as recessive in case of a 29-bit frame. CAN controllers that support extended frame format messages are also able to send and receive messages in CAN base frame format. All frames begin with a start-of-frame (SOF) bit that denotes the start of the frame transmission.

CAN has four frame types:

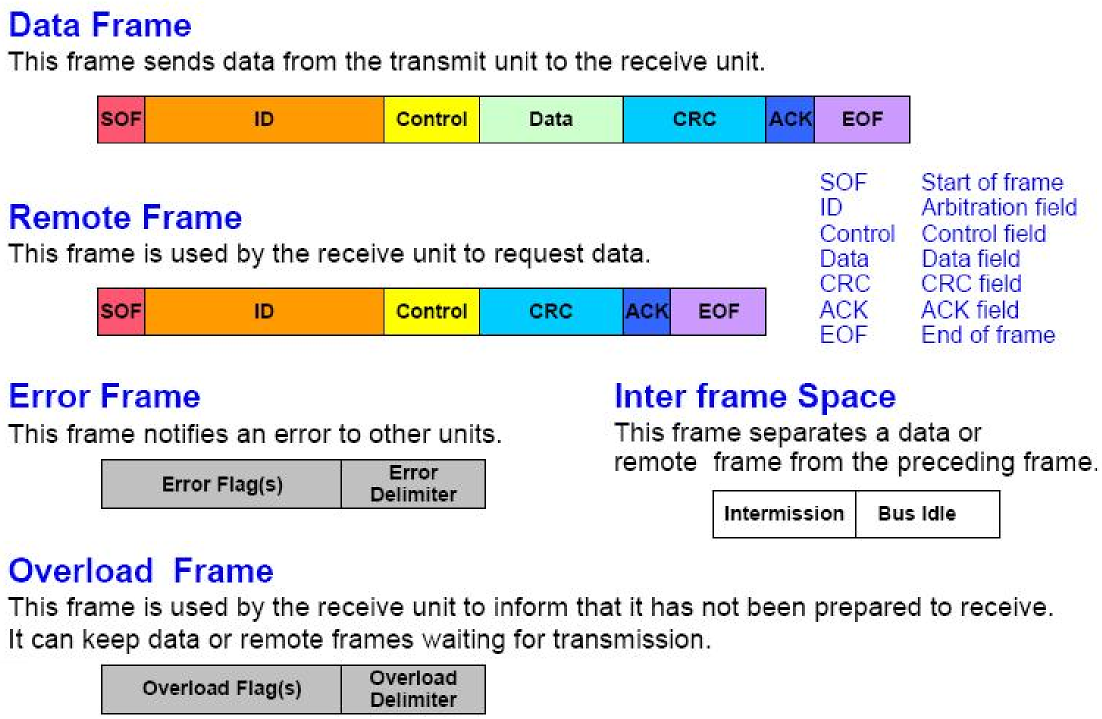

Figure 3.4 Types of CAN frame

Data frame: a frame containing node data for transmission

Remote frame: a frame requesting the transmission of a specific identifier

Error frame: a frame transmitted by any node detecting an error

Overload frame: a frame to inject a delay between data and/or remote frames [5]

- **Data frame**

The data frame is the only frame for actual data transmission. There are two message formats:

Base frame format: with 11 identifier bits

Extended frame format: with 29 identifier bits

The CAN standard requires the implementation must accept the base frame format and may accept the extended frame format, but must tolerate the extended frame format.

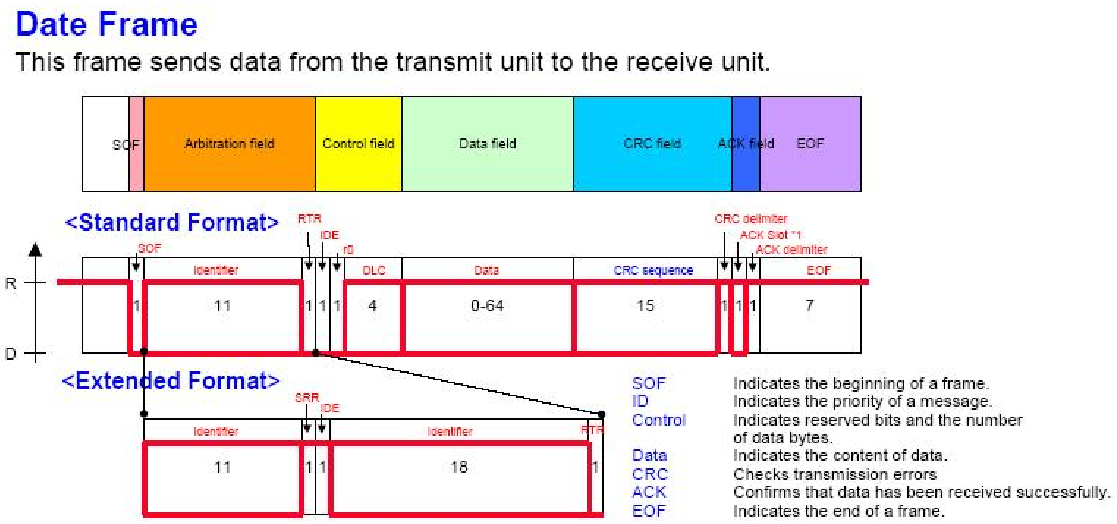

Figure 3.5 CAN data frame

Base frame format [5]

The frame format is as follows:

Field name	Length (bits)	Purpose
Start-of-frame	1	Denotes the start of frame transmission
Identifier	11	A (unique) identifier for the data
Remote transmission request (RTR)	1	Dominant (0) (see Remote Frame below)
Identifier extension bit (IDE)	1	Must be dominant (0)Optional
Reserved bit (r0)	1	Reserved bit (it must be set to dominant (0), but accepted as either dominant or recessive)
Data length code (DLC)*	4	Number of bytes of data (0-8 bytes)
Data field	0-8 bytes	Data to be transmitted (length dictated by DLC field)

CRC	15	Cyclic Redundancy Check
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)
End-of-frame (EOF)	7	Must be recessive (1)

One restriction placed on the identifier is that the first seven bits cannot be all recessive bits. (I.e., the 16 identifiers 1111111xxxx are invalid.)

Extended frame format

The frame format is as follows:

Field name	Length (bits)	Purpose
Start-of-frame	1	Denotes the start of frame transmission
Identifier A	11	First part of the (unique) identifier for the data
Substitute remote request (SRR)	1	Must be recessive (1)Optional
Identifier extension bit (IDE)	1	Must be recessive (1)Optional
Identifier B	18	Second part of the (unique) identifier for the data
Remote transmission request (RTR)	1	Must be dominant (0)
Reserved bits (r0, r1)	2	Reserved bits (it must be set dominant (0), but

		accepted as either dominant or recessive)
Data length code (DLC)*	4	Number of bytes of data (0-8 bytes)
Data field	0-8 bytes	Data to be transmitted (length dictated by DLC field)
CRC	15	Cyclic redundancy check
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)
End-of-frame (EOF)	7	Must be recessive (1)

The two identifier fields (A & B) combined form a 29-bit identifier.

* It is physically possible for a value between 9-15 to be transmitted in the 4-bit DLC, although the data is still limited to 8 bytes. Certain controllers allow the transmission and/or reception of a DLC greater than 8, but the actual data length is always limited to 8 bytes.

- **Remote frame**

Generally data transmission is performed on an autonomous basis with the data source node (e.g. a sensor) sending out a Data Frame. It is also possible, however, for a destination node to request the data from the source by sending a Remote Frame.

There are 2 differences between a Data Frame and a Remote Frame. Firstly the RTR-bit is transmitted as a dominant bit in the Data Frame and secondly in the Remote Frame there is no Data Field.

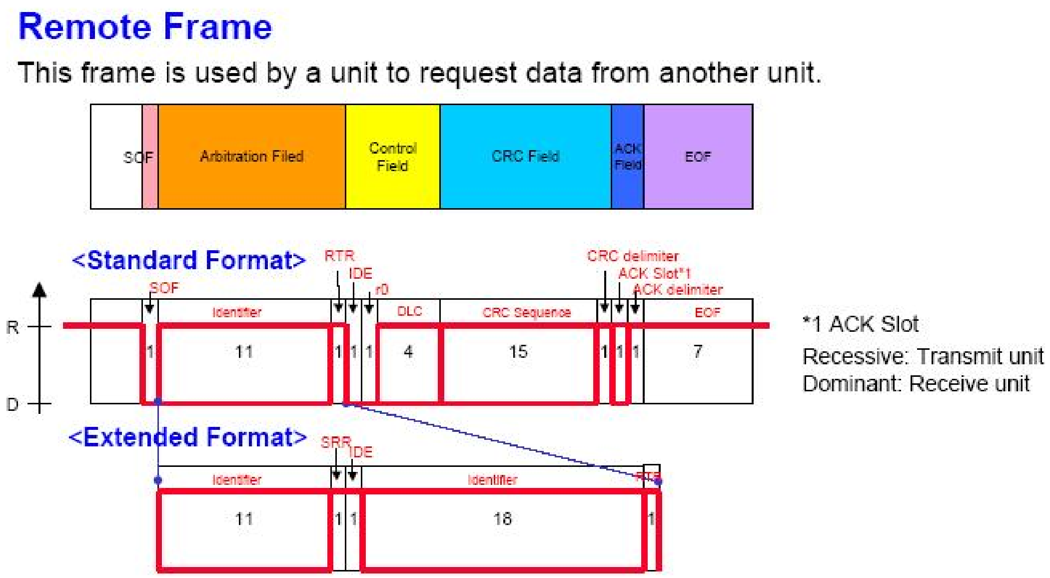

Figure 3.6 CAN remote frame

i.e.

RTR = 0 ; DOMINANT in data frame

RTR = 1 ; RECESSIVE in remote frame

In the very unlikely event of a Data Frame and a Remote Frame with the same identifier being transmitted at the same time, the Data Frame wins arbitration due to the dominant RTR bit following the identifier. In this way, the node that transmitted the Remote Frame receives the desired data immediately.

- **Error frame**

Error frame consists of two different fields

The first field is given by the superposition of ERROR FLAGS contributed from different stations. The following second field is the ERROR DELIMITER.

There are two types of error flags

Active Error Flag

Transmitted by a node detecting an error on the network that is in error state "error active".

Passive Error Flag

Transmitted by a node detecting an active error frame on the network that is in error state "error passive".

- **Overload frame**

The overload frame contains the two bit fields Overload Flag and Overload Delimiter. There are two kinds of overload conditions that can lead to the transmission of an overload flag:

1. The internal conditions of a receiver, which requires a delay of the next data frame or remote frame.
2. Detection of a dominant bit during intermission.

The start of an overload frame due to case 1 is only allowed to be started at the first bit time of an expected intermission, whereas overload frames due to case 2 start one bit after detecting the dominant bit. Overload Flag consists of six dominant bits. The overall form corresponds to that of the active error flag. The overload flag's form destroys the fixed form of the intermission field. As a consequence, all other stations also detect an overload condition and on their part start transmission of an overload flag. Overload Delimiter consists of eight recessive bits. The overload delimiter is of the same form as the error delimiter.

3.2.4 CAN data transmission

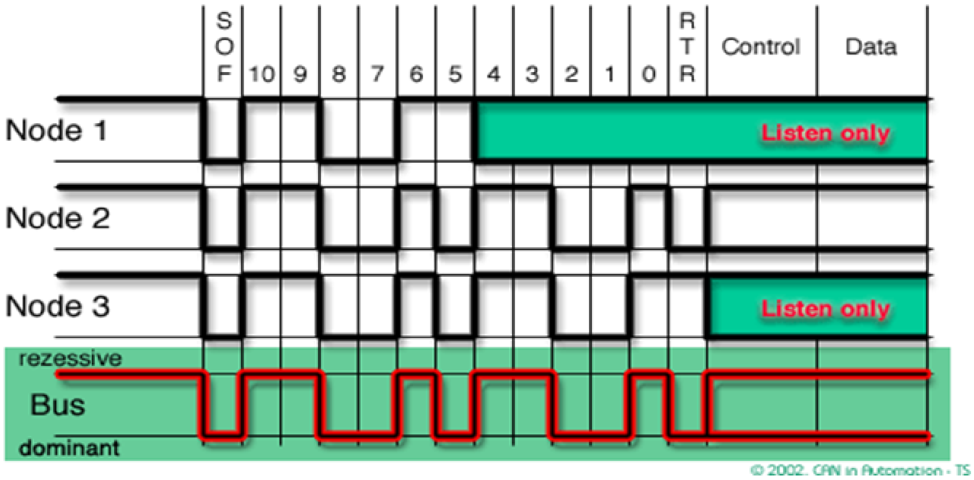

Figure 3.7 CAN bus signal priority

CAN features an automatic 'arbitration free' transmission. A CAN message that is transmitted with highest priority will 'win' the arbitration, and the node transmitting the lower priority message will sense this and back off and wait. [5]

This is achieved by CAN transmitting data through a binary model of "dominant" bits and "recessive" bits where dominant is a logical 0 and recessive is a logical 1. This means open collector, or 'wired or' physical implementation of the bus (but since dominant is 0 this is sometimes referred to as wired-AND). If one node transmits a dominant bit and another node transmits a recessive bit then the dominant bit "wins".

So, if you are transmitting a recessive bit, and someone sends a dominant bit, you see a dominant bit, and you know there was a collision. A dominant bit is asserted by creating a voltage across the wires while a recessive bit is simply not asserted on the bus. If any node sets a voltage difference, all nodes will see it. Thus there is no delay to the higher priority messages, and the node transmitting the lower priority message automatically attempts to re-transmit 6 bit clocks after the end of the dominant message.

When used with a differential bus, a Carrier Sense Multiple Access/Bitwise Arbitration (CSMA/BA) scheme is often implemented: if two or more devices start transmitting at the same time, there is a priority based arbitration scheme to decide which one will be granted permission to continue transmitting. The CAN solution to this is prioritized arbitration (and for the dominant message delay free), making CAN very suitable for real time prioritized communications systems.

During arbitration, each transmitting node monitors the bus state and compares the received bit with the transmitted bit. If a dominant bit is received when a recessive bit is transmitted then the node stops transmitting (i.e., it lost arbitration). Arbitration is performed during the transmission of the identifier field. Each node starting to transmit at the same time sends an ID with dominant as binary 0, starting from the high bit. As soon as their ID is a larger number (lower priority) they'll be sending 1 (recessive) and see 0 (dominant), so they back off. At the end of ID transmission, all nodes but one have backed off, and the highest priority message gets through unimpeded.

3.3 CAN bus interface

The P-bus has a data transfer rate of 500 kbits/s and the I-bus of 33 kbits/s. The data rate on the Pbus is high to allow the powertrain systems access to information with the shortest possible delay.

The control modules send out information on the bus whenever the information changes. As a safety precaution, the information is also sent at regular intervals irrespective of whether or not it is new.

The time between two transmissions depends on the information being transmitted and varies between 10 milliseconds (engine torque) and 10 second. When a control module sends information, the other control modules listen regardless of whether or not they use the information.

With CAN bus interface (Figure 3.6), we can send message to CAN network, or receive CAN message on the network. This interface can send one message at a time or several messages at a time depend on what user need.

Figure 3.8 CAN bus interface

3.4 CAN applications

3.4.1 application

A modern automobile may have as many as 50 electronic control units (ECU) for various subsystems. Typically the biggest processor is the engine control unit, which is also referred to as "ECU" in the context of automobiles; others are used for transmission, airbags, antilock braking, cruise control, audio systems, windows, doors, mirror adjustment, etc. Some of these form independent subsystems, but communications among others are essential. A subsystem may need to control actuators or receive feedback from sensors. The CAN standard was devised to fill this need.

The CAN bus may be used in vehicles to connect engine control unit and transmission, or to connect the door locks, climate control, seat control, etc. Today the CAN bus is also used as a fieldbus in general automation environments, primarily due to the low cost of some CAN Controllers and processors.

Bosch holds patents on the technology, and manufacturers of CAN-compatible microprocessors pay licence fees to Bosch, which is normally passed on to the customer in the price of the chip. Manufacturers of products with custom ASICs or FPGAs containing CAN-compatible modules, may need to pay a fee for the CAN Protocol License.

CHAPTER 4

Saab 95 SID simulation

4.1 SID introduction

SID (SAAB Information Display) is a screen (Figure 4.1) on SAAB 95. It has so many modes to display different information, such as speed, temperature, fuel economy, tire pressure, time…etc.

Figure 4.1 SID screen on SAAB 95

4.2 SID function

Transmission on SID is based on low speed CAN-bus. It shows information of other devices, and it also sends CAN messages to the others. SID can show strings and make warning sound to inform driver what serious happen on the car to prevent accident.

4.3 SID application

0x328 SID audio text (must be sent with ID: 0x348) [16]

A group of three messages are sent with an interval of 1 second and if a value changes. Messages are sent with about 10 milliseconds apart. The CHANGED bit in the ROW byte will be set if information changes. The two bits ORD0 and ORD1 in the ORDER byte are for sequence numbering. A new message group starts with the NEW bit set and both ORD0 and ORD1 bits. On the second message the NEW and ORD1 bits are cleared and the third message has also the ORD0 bit cleared. So the ORDER byte will be 42, 01 and 00 for the three sequential messages. The ROW byte tells to which row of the SID the text will be displayed on. The byte can have a value of 2 or 1, but in these audio text messages the row number is always 2. The TEXT4...TEXT0 bytes are plain ASCII coded characters that will be displayed on the SID.

Note that the last message will contain only one normal character in the TEXT4 byte. The TEXT3 byte should contain a integer value that will be shown as a small number to indicate the selected radio station number. The last TEXT2...TEXT0 bytes are always zeros.

ID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
328h	order		row	Text4	Text3	Text2	Text1	Text0
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
order		new					Ord1	Ord0
row	changed						Row1	Row0

Example message group:

42 96 02 44 4F 4F 52 20 DOOR

01 96 02 4F 50 45 4E 20 OPEN

00 96 02 20 00 00 00 00

The text "DOOR OPEN " will be displayed on the SID display.

Users can change text by changing the last 5 bytes of each message.

348h SID audio text control

Message is sent with an interval of 1 second. The normal value (i.e. don't display text) of byte STATUS is FFh. When a 328h SID audio text message group is sent, the STATUS byte is changed to 04h and immediately after that to 05h for the duration of the messages. After the 328h messages, the status byte returns to FFh.

Notice that the control messages use a 20h increment relative to the actual text messages in their IDs. Bytes 0 and 3 probably have something to do with priority (which text is displayed if several texts should be displayed at the same time).

ID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
348h	11	02	status	19	00	00	00	00

Example message:

11 02 FF 19 00 00 00 00

SID should not display the text provided with message 328h.

11 02 05 19 00 00 00 00

SID should display the text provided with message 328h, and the text will disappear after signal finish.

11 01 05 19 00 00 00 00

SID should display the text provided with message 328h, and the text will still be on the screen even signal finish.

434h – SID warning sound

Byte 0 is state, 00 mean no message is change, when a message is sent, Byte 0 turn to 80. Byte 1 is warning sound, different bit means different sound.

Example message:

00 00 00 00 00 00 00 00

No message is sent

Example message:

80 40 00 00 00 00 00 00

SID has a warning sound

4.4 SID simulation result

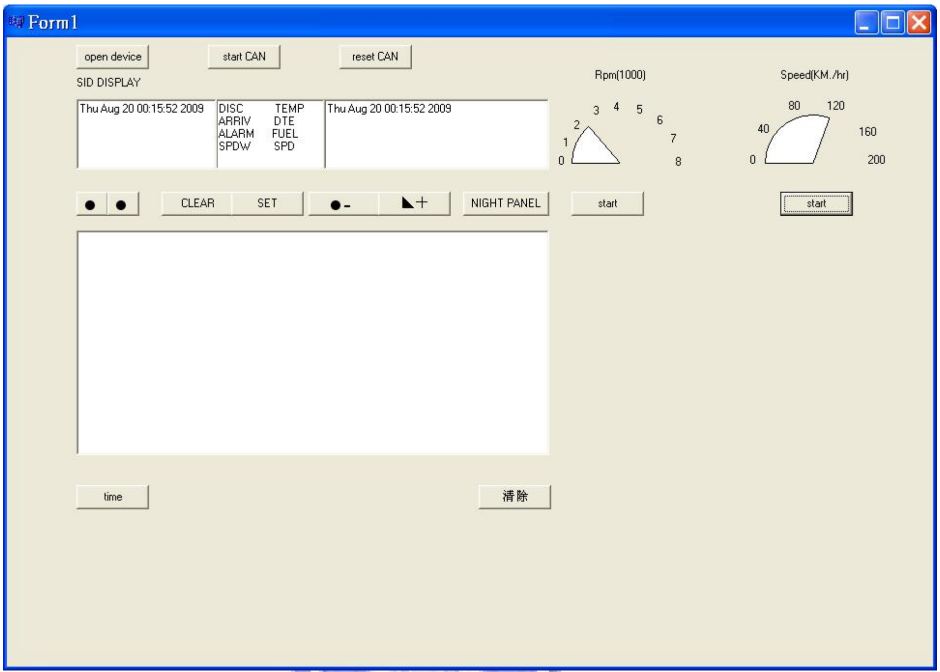

Figure 4.2 SID simulation on SAAB 95

There are two speed of CAN-bus signal, one is high speed P-bus signal which baud rate is 500Kbits/s, the other is low speed I-bus which baud rate is 47.059Kbits/s. Some devices use high speed channel, some use low speed channel, and others have both high speed and low speed channel.

SID simulation result is showed in Figure 4.2. It can show vehicle information on the screen. For example, RPM, vehicle speed, cooler temperature, tire pressure, fuel economy, current time…etc.

For example, if we get ID:0x1A0 data: 1A 08 98 32 00 00 64 00

(0898)16 = (2200)10

2200 * 180 / 8000 = 49.5°

In this example, if we want to know vehicle RPM signal, we receive message with ID: 0x1A0 Engine information. Calculate Byte 1 and Byte 2 value from 16-bit into decimal to get the angle 49.5°, so RPM pointer will move to angel 49.5°.

1A0h - Engine information

Message is sent with an interval of 10 milliseconds. The ENGSTOPPED bit in the STATUS byte indicates if the engine is running or stopped. Engine RPM is shown as a 16-bit value (RPM1 and RPM0). MAP shows the Manifold Absolute Pressure in kilopascals. THROTTLE byte is the throttle pedal position, with a value range of 0...100 percent (0...64hex). MAF is the reading from Mass Air Flow sensor.

ID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
1A0h	status	Rpm 1	Rpm 0	map		throttle		maf
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
status	changed				Eng stopped			

For example, if we get ID:0x2F0 data: 00 07 30 00 00 00 00 00

(0730)16 = (1150)10

1150 / 10 = 115 KM/H

115 * 180 / 200 = 103.5°

In this example, if we want to know vehicle RPM signal, we receive message with ID: 0x2F0 Vehicle speed. Calculate Byte 1 and Byte 2 value from 16-bit into decimal to get the angle 103.5°, so speed pointer will move to angel 103.5°.

2F0h - Vehicle speed

Message is sent with an interval of 20 milliseconds. Vehicle speed is indicated by the 16-bit value (bytes SPEED1 and SPEED0). You will need to divide the value by 10 to get the right scaling, kilometers per hour. The byte 3 seems to be 80h all the time.

ID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
2F0h		Speed1	Speed0					

CHAPTER 5

CAN-MOST interface

Purpose of this chapter is to combine two networks (CAN-MOST). This program can control MOST system and send CAN signal.

5.1 Control MOST system by CAN-MOST interface

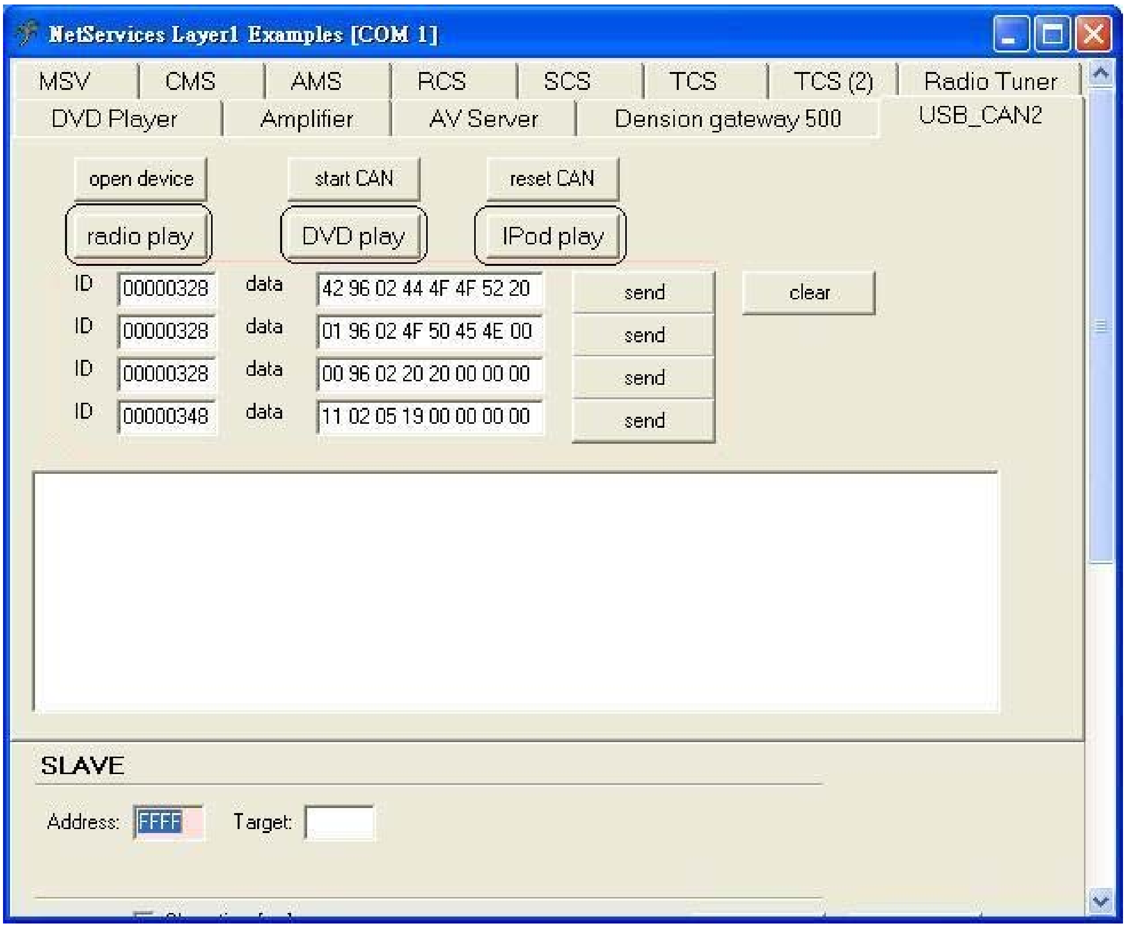

Figure 5.1 USBCAN2 under MOST Network (control MOST device)

Press radio play button can listen to the radio on the MOST network; Press DVD play button can watch DVD movies; and Press IPod play button can listen to the music in IPod. As show in Figure 5.1.

5.2 Send CAN messages by CAN-MOST interface

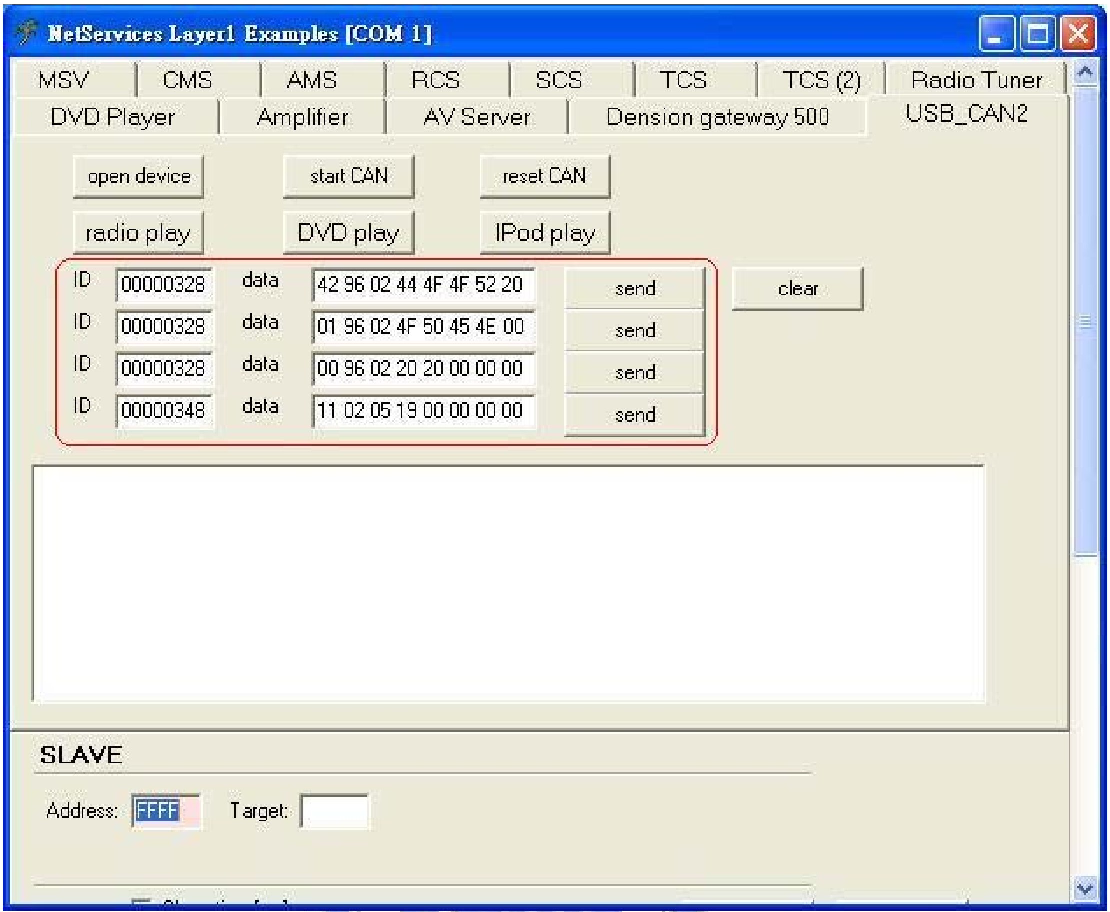

Figure 5.2 USBCAN2 under MOST Network (send CAN messages)

Sending CAN messages to control CAN devices on CAN network by CAN-MOST interface. For example, after sending these four messages (as show in Figure 5.2), SID will show the string "DOOR OPEN" on the screen.

Figure 5.3 SID screen on SAAB 95

5.3 Application of CAN-MOST interface

Due to safety consideration, when vehicle speed reaches 20KM/H, DVD player will shout down. For example, when we receive CAN message ID: 0x2F0 Vehicle speed. If the value of Byte1 and Byte2 > 00C8(=200_{10}) , MOST DVD player will stop.

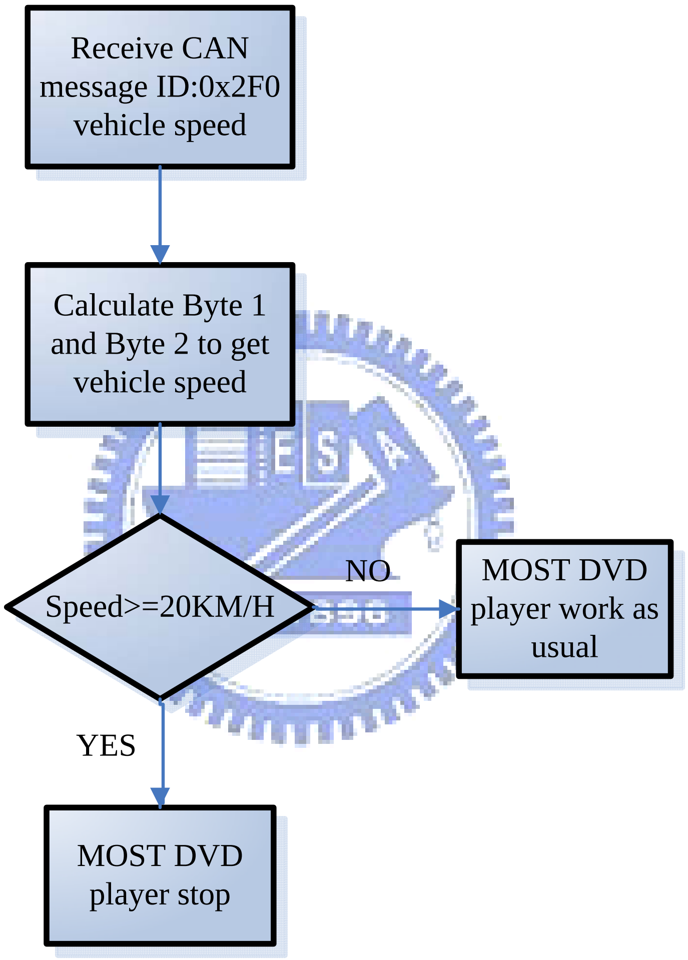

Figure 5.4 Flowchart of MOST DVD player during driving

Due to safety consideration, when vehicle speed reaches 60KM/H, MOST Amplifier will tune up the volume. when vehicle speed reaches 100KM/H, MOST Amplifier will tune up the volume again. For example, when we receive CAN message ID: 0x2F0 Vehicle speed. If the value of Byte1 and Byte2 > 258(=600_{10}) , MOST Amplifier will tune up the volume. If the value of Byte1 and Byte2 > 3E8(=1000_{10}) , MOST Amplifier will tune up the volume again.

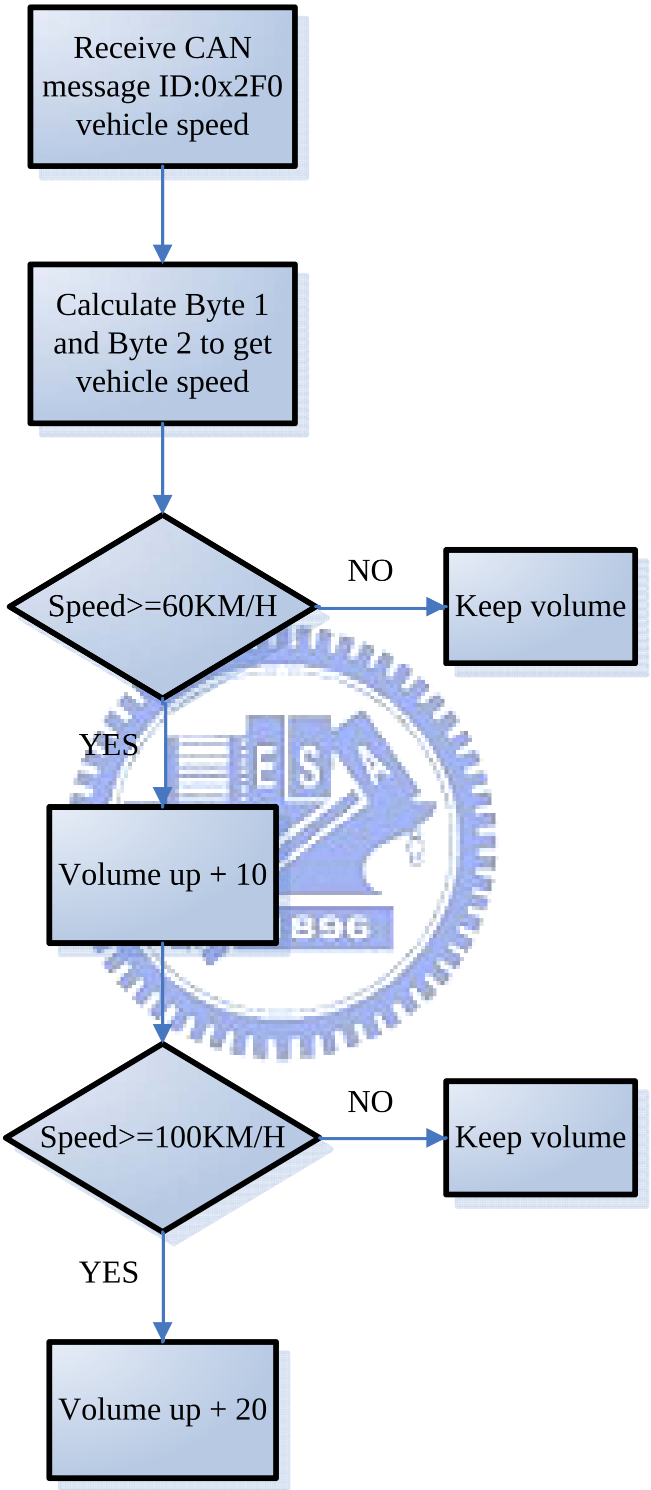

Figure 5.5 Flowchart of MOST Amplifier during driving

2F0h - Vehicle speed

Message is sent with an interval of 20 milliseconds. Vehicle speed is indicated by the 16-bit value (bytes SPEED1 and SPEED0). You will need to divide the value by 10 to get the right scaling, kilometers per hour. The byte 3 seems to be 80h all the time.

ID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
2F0h		Speed1	Speed0					

Due to safety consideration, when coolant temperature is too high, SID display will show warning message (high coolant temperature), and make a warning sound to inform drivers. As show in Figure 5.6.

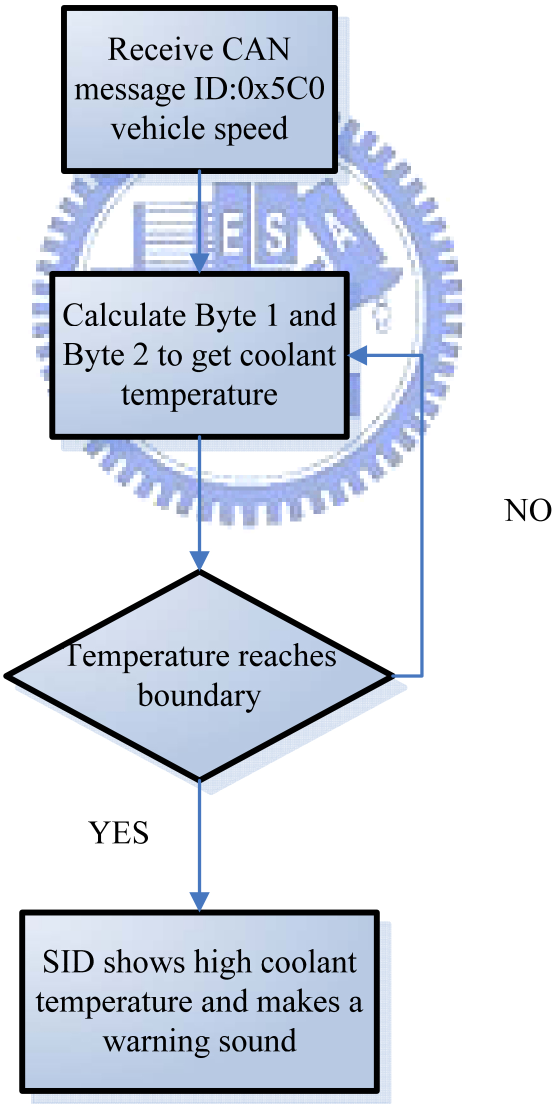

Figure 5.6 Flowchart of coolant temperature during driving

5C0h - Coolant temperature, air pressure

Message is sent with an interval of 1 second. Temperature is reported with a 8-bit byte. In order to get the correct coolant temperature, the value must be subtracted with 40. This is done to encode negative temperatures. So a value of 58 (3Ah) is in fact +18 degrees Celsius and on the other hand a value of 29 (1Dh) would give an temperature of -11 degrees Celsius. The 16-bit value combined from PRES1 and PRES0 gives the Ambient air pressure in hehtopascals [hPA].

ID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
5C0h		coolant	coolant	Pres1	Pres0			

Example message:

00 6A 6A 03 F9 00 00 00

Coolant temperature is 66 degrees Celsius and air pressure is 1017 hPa.

CHAPTER 6

Conclusion

CAN-MOST interface can receive, send and analyze CAN messages. The data received by the program will be analyzed by user. Depending on these messages, user will analyze if those are important. Otherwise, they will be ignored.

The contribution of my MS thesis is CAN-MOST interface. We have to control MOST devices and CAN devices individually before the program is finished. The main purpose of this program is to control MOST devices and CAN devices in the same time without opening two programs. Now we can control all devices by this program. For example, play DVDs, listen to IPod, listen to radio, control SID…etc.

Future works.

In the future, as CAN-MOST interface grow mature, LIN bus can be added into this program. After the combination of these three networks (LIN, CAN, MOST), we can reduce cost for modern automobile industry. By further research, this program will be more complete and reliable.

REFERENCES

[1] Jun-Ying Huang, "Design of Audio/Video Sever under MOST (Media Oriented System Transport) Network" National Chiao Tung University, Master thesis, Nov. 2006.

[2] Oasis SiliconSystems, "MOST NetServices Layer 1 User Manual/Specification Rev. 1.10.x," Jan 2004.

[3] MOST Cooperation website, http://www.mostcooperation.com/home/index.html

[4] SMSC website, http://www.smsc-ais.com/AIS/

[5] CAN in Automation http://www.can-cia.org/

[6] "CAN History". CAN in Automation. http://www.can-cia.de/index.php?id=161

[7] SAE international http://www.sae.org/servlets/index

[8] OS8104 datasheet

[9] MOST Cooperation, "MOST Specification Framework Rev 1.1," 1999.

[10] MOST Cooperation, "MOST Specification Rev 2.5," Oct. 2006.

[11] SMSC, "MOST PCI Interfaces Cookbook V1.0.x User Manual," Dec. 2006.

[12] Oasis SiliconSystems, "RadioTuner 4 MOST V1.0.0 User Manual," August 2003.

[13] Oasis SiliconSystems, "DVDPlayer 4 MOST V1.2.0 User Manual," Oct. 2003.

[14] Oasis SiliconSystems, "Amplifier 4 MOST Version: 2.0 User Manual," July 2003.

[15] ISO 11898 datasheet

[16] Tomi Liljemark http://pikkupossu.1g.fi/tomi/projects/projects.html