Development of Communication Manager Module for Automotive Platform Software based on AUTOSAR 4.0

Neethu Mary Judy
M. Tech(Embedded Systems)
Sree Buddha College of Engineering
Pattoor, Kerala
India

Jayaraj V. S
Department of ECE
Sree Buddha College of Engineering
Pattoor, Kerala
India

Kuruvilla Jose
Specialist
Automotive Electronics
Tata Elxsi
Trivandrum, Kerala
India

***Abstract*—Automotive industry is growing rapidly. This rapid growth has increased the challenges in different forms including complexity. AUTomotive Open System Architecture (AUTOSAR) provides a standardized automotive software platform. Communication Manager (ComM) module in BSW (Basic Software) layer is responsible for the control of communication services. This paper is about the development of ComM module in AUTOSAR 4.0. The functionality of ComM module is obtained by realizing specific Application Programming Interfaces (APIs).**

***Keywords*—AUTOSAR, ComM, API, ECU**

I. INTRODUCTION

Development in technology and increase in demands led to tremendous growth in automotive industries. The ECUs in vehicles serve different functionality. Since ECUs are delivered by different vendors, the automobile manufacturers experienced difficulties in terms of complexity, diagnostics, inter ECU communication etc. So there arose a need for standard software in ECUs. AUTOSAR is standardized and an open automobile platform software. The goal of AUTOSAR is to have a transition from customer specific software to functional specific software. It is hardware independent.

II. AUTOSAR SOFTWARE ARCHITECTURE

A basic design concept of the AUTOSAR software stack is the separation between application and infrastructure. AUTOSAR standard follows a layered architecture, which provides modularity concept. The software architecture contains three basic layers: Application layer, RTE (Run Time Environment) layer and Basic Software (BSW) layer. Fig. 1 shows the software layers in AUTOSAR architecture.

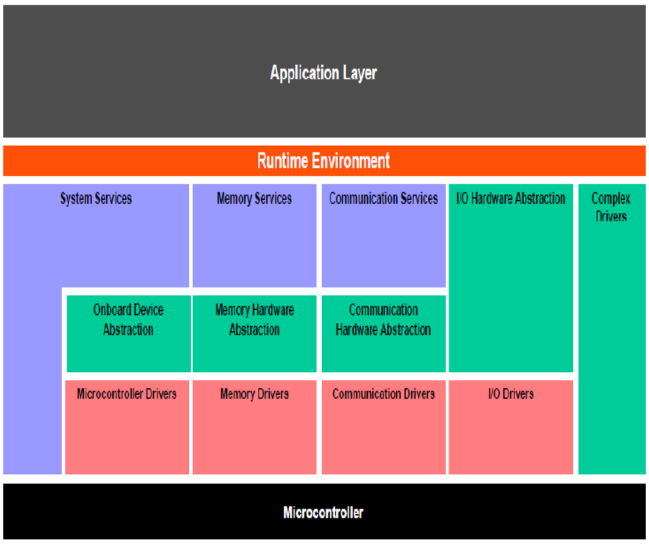


Fig.1. Software layers of AUTOSAR

RTE allow the AUTOSAR software components to be independent from the mapping to the specific ECUs. BSW is further divided into Services, ECU Abstraction, Micro controller Abstraction and complex drivers. Being the lowest software layer, Micro controller Abstraction layer has direct access to the micro controller and internal peripherals. ECU Abstraction layer interfaces the drivers of Micro controller Abstraction layer. The highest layer in Basic Software layer is the System Service layer which serves a number of services. These services include vehicle communication management, ECU state management, diagnostics services, memory management etc. The task of complex drivers is to provide the possibility to integrate special purpose functionality.

III. COMMUNICATION MANAGER MODULE

Communication Manager (ComM) module acts as a resource manager. It is responsible for managing the underlying communication services. Functions of ComM include allocation of necessary resources for the requested communication mode, switches to a communication mode as requested by the user, implement channel state machine for every channel to control more than one communication channel of an ECU etc.

A. Communication Modes

The ComM module provides three different communication modes.

1. COMM_NO_COMMUNICATION
2. COMM_FULL_COMMUNICATION
3. COMM_SILENT_COMMUNICATION

The COMM_FULL_COMMUNICATION state has the following sub states:

- COMM_FULL_COM_NETWORK_REQUESTED
- COMM_FULL_COM_READY_SLEEP

The COMM_NO_COMMUNICATION state has the following sub states:

- COMM_NO_COM_REQUEST_PENDING
- COMM_NO_COM_NO_PENDING_REQUEST

A user cannot request for COMM_SILENT_COMMUNICATION. The default state of a communication channel is COMM_NO_COMMUNICATION with sub state COMM_NO_COM_NO_PENDING_REQUEST. ComM channel will allow communication only if CommunicationAllowed flag is TRUE in COMM_NO_COM_REQUEST_PENDING.

IV. DESIGN

ComM module is developed by coding all the APIs realized by it for satisfying its functional specification. Communication (Com) module can transmit and receive messages only if ComM module makes the channel in COMM_FULL_COMMUNICATION state. COMM_NO_COMMUNICATION mode does not support transmission and reception of messages. For the design of ComM module, High Level Design (HLD) and Low Level Design (LLD) were carried out. HLD includes the identification of the APIs needed for the implementation of the ComM module and LLD gives a detailed idea and design of each APIs identified in HLD. Both HLD and LLD are done using the tool Enterprise Architect Version 9.3.

A. High Level design of ComM

Fig. 2 specifies the dependency diagram in which dependencies to other modules is shown. The Communication Manager Module requests the communication capabilities, requested from the users, from the Bus State Manager Modules.

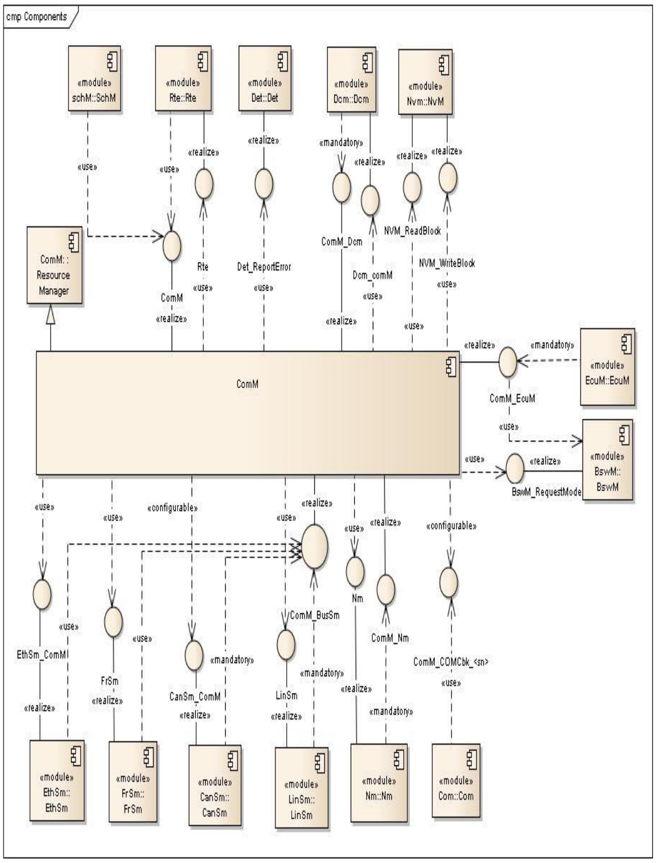


Fig. 2. High level design of ComM

ComM module is initialized by EcuM-Fixed. All validation of wake up events is done by both EcuM (Fixed and Flex) and indicated to ComM module. If COMM_FULL_COMMUNICATION mode is requested by DCM, then only Diagnostics shall be performed. A communication mode is requested from the CAN state manager by ComM and the bus state is mapped to this mode by the CAN state manager.

B. Low Level design of ComM

The Low Level Design involves the activity diagram of each of the APIs to realize the module. Some of the LLD of APIs is shown below.

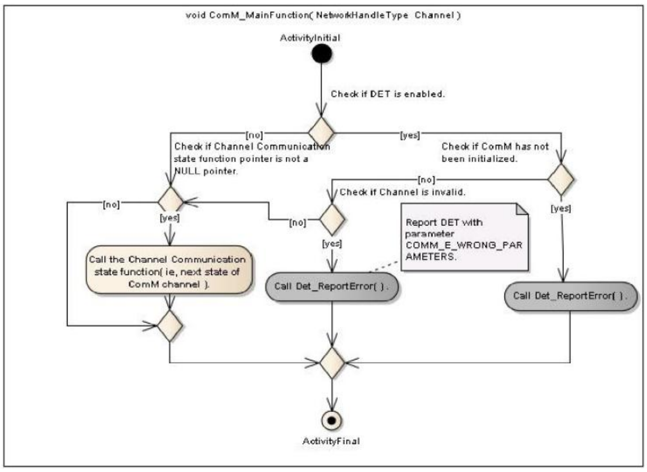

void ComM_MainFunction(NetworkHandleType Channel)
ActivityInitial
Check if DET is enabled.
[no]
[yes]
Check if Channel Communication state function pointer is not a NULL pointer.
Check if ComM has not been initialized.
[no]
Check if Channel is invalid.
[no]
[yes]
[no]
[yes]
[yes]
Report DET with parameter COMM_E_WRONG_PARAMETERS.
Call the Channel Communication state function(ie, next state of ComM channel).
Call Det_ReportError( ).
Call Det_ReportError( ).
ActivityFinal

Fig. 3. Flow chart of ComM_Mainfuncton()

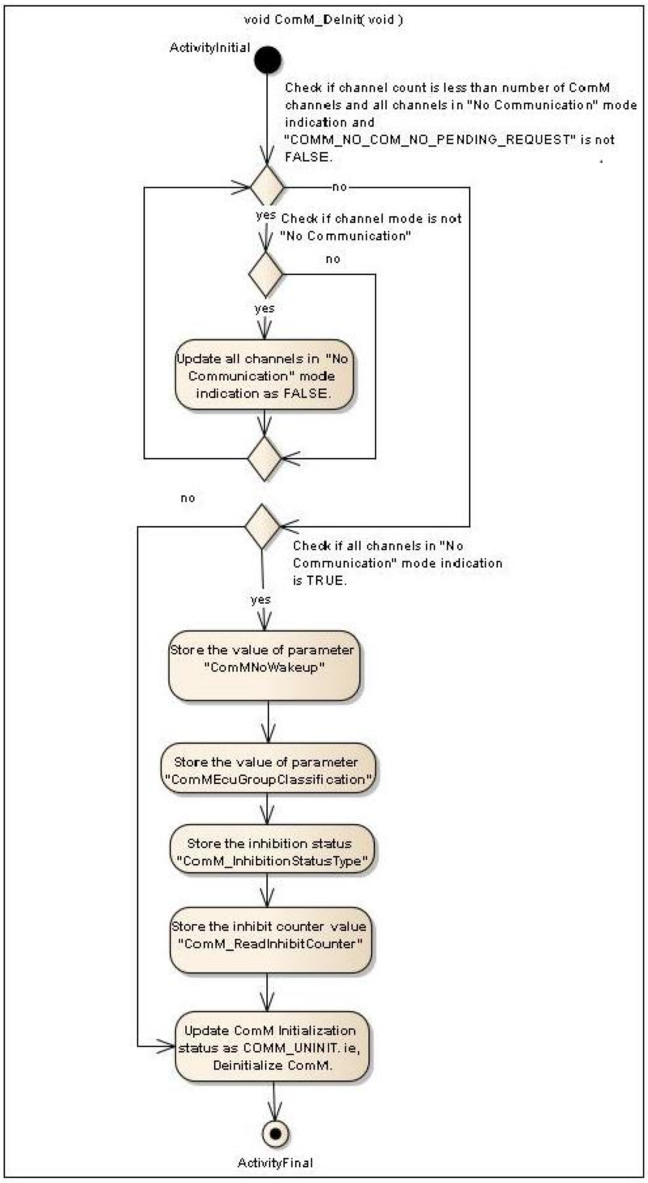

void ComM_DeInit(void)
ActivityInitial
Check if channel count is less than number of ComM channels and all channels in "No Communication" mode indication and "COMM_NO_COM_NO_PENDING_REQUEST" is not FALSE.
no
yes
Check if channel mode is not "No Communication"
no
yes
Update all channels in "No Communication" mode indication as FALSE.
no
Check if all channels in "No Communication" mode indication is TRUE.
yes
Store the value of parameter "ComMNoWakeup"
Store the value of parameter "ComMEcuGroupClassification"
Store the inhibition status "ComM_InhibitionStatusType"
Store the inhibit counter value "ComM_ReadInhibitCounter"
Update ComM Initialization status as COMM_UNINIT. ie, Deinitialize ComM.
ActivityFinal

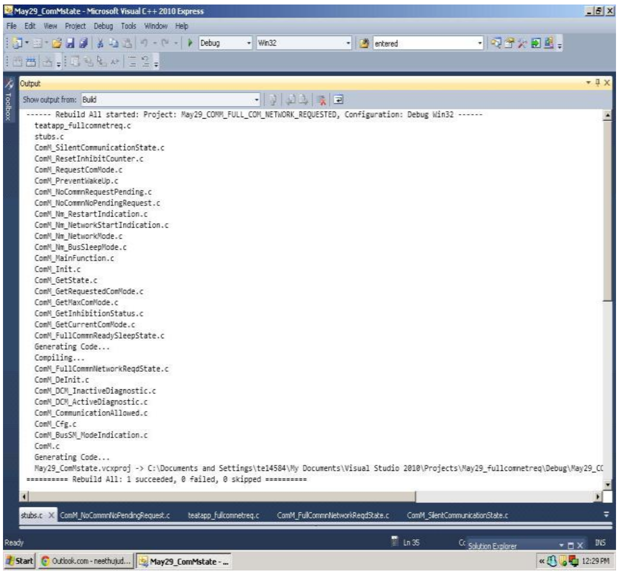
Fig. 4. Flow chart of ComM_DeInit()

V. Implementation And Testing

Coding for the development of ComM module is done using ‘C’ language. The file structure contains header files and source files. The header files include ComM.h, ComM_Nm.h, ComM_EcuMBswM.h, ComM_Dcm.h, ComM_BusSm.h and ComM_Cfg.h. These header files are in AUTOSAR standard.

ComM.h contains the declarations of the ComM APIs and type definitions of data types. ComM_Nm.h, ComM_EcuMBswM.h, ComM_Dcm.h, ComM_BusSm.h include the ComM callback declarations. The ComM_Cfg.h and ComM.c are the configuration and source file spectively. The later defines all the APIs realized by ComM module.

Coding is performed in Visual C++ 2010 express edition. It is successfully compiled and built without any errors. Snap shot of the code successfully built in Visual C++ 2010 express edition is shown in Fig. 5.



```
------ Rebuild All started: Project: May29_COMM_FULL_COM_NETWORK_REQUESTED, Configuration: Debug Win32 ------
  teatapp_fullcomnetreq.c
  stubs.c
  ComM_SilentCommunicationState.c
  ComM_ResetInhibitCounter.c
  ComM_RequestComMode.c
  ComM_PreventWakeUp.c
  ComM_NoCommnRequestPending.c
  ComM_NoCommnNoPendingRequest.c
  ComM_Nm_RestartIndication.c
  ComM_Nm_NetworkStartIndication.c
  ComM_Nm_NetworkMode.c
  ComM_Nm_BusSleepMode.c
  ComM_MainFunction.c
  ComM_Init.c
  ComM_GetState.c
  ComM_GetRequestedComMode.c
  ComM_GetMaxComMode.c
  ComM_GetInhibitionStatus.c
  ComM_GetCurrentComMode.c
  ComM_FullCommnReadySleepState.c
  Generating Code...
  Compiling...
  ComM_FullCommnNetworkReqdState.c
  ComM_DeInit.c
  ComM_DCM_InactiveDiagnostic.c
  ComM_DCM_ActiveDiagnostic.c
  ComM_CommunicationAllowed.c
  ComM_Cfg.c
  ComM_BusSM_ModeIndication.c
  ComM.c
  Generating Code...
  May29_ComMstate.vcxproj -> C:\Documents and Settings\te14584\My Documents\Visual Studio 2010\Projects\May29_fullcomnetreq\Debug\May29_CC
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
```

Fig. 5. Snap shot of code build in VC++ 2010 express edition

Testing of ComM module is done in two phases:

- Unit testing
- Integrated testing

In unit testing the functionality of all APIs are checked individually. Testing is done in Visual C++ in which test application was written for each API and functionality verified. Unit testing helps in easy debugging. Integrated testing deals validation of overall functionality of the application specified. Different BSW modules were integrated and validation was performed on MPC 5668G Evaluation board.

VI. Results

In unit testing all APIs are functionally verified. The integration testing validated functional performance and reliability requirements placed on major design items. It involves the verification of basic functionality of application and BSW modules after integration. Fig.6 shows the evaluation board used for integrated testing. Fig .7 shows the integrated test set up

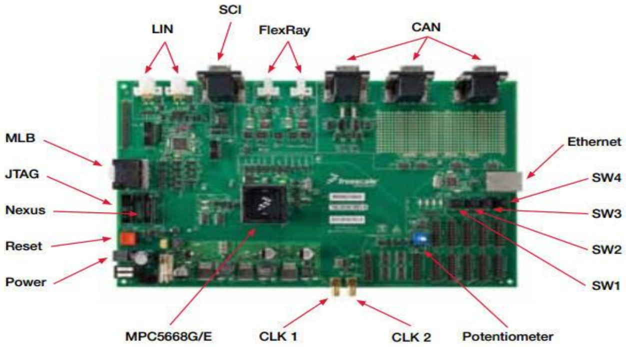


Fig. 6. MPC5668G evaluation board

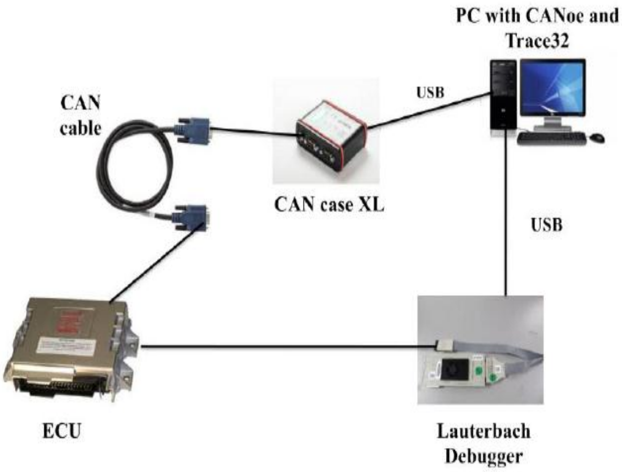


Fig. 7. Integration Test Setup

ACKNOWLEDGMENT

This research was supported by ECE Dept. of Sree Buddha College of Engineering, Pattoor, kerala. The authors would also like to thank Tata Elxsi, Trivandrum for providing insight and expertise that greatly assisted the research.

REFERENCES

[1] AUTOSAR, Specification of Communication Manager V4.0.0 R4.0
Rev 3.
[Online].Available.http://www.autosar.org/

[2] AUTOSAR, Specification of Communication Manager V2.1.0 R3.0 Rev 7.
[Online]. Available.http://www.autosar.org/

[3] SAE, International Society of Automotive Engineers.
[Online]. Available.http://automobile. sae.org/

[4] S. Furst, "AUTOSAR for Safety-related Systems: Objectives, Approach and Status", in Second IEEE Conference on Automotive Electronics, London, United Kingdom, March 2006.

[5] G. Leen and D.Heffernan. "Expanding Automotive Electronic Systems", IEEE Computer, 35(1), January 2002, pp. 88–93.

[6] MPC5668x Microcontroller Reference Manual, Document Number: MPC5668XRM Rev.4 01/2011