International Journal of Enhanced Research in Science Technology & Engineering, ISSN: 2319-7463
Vol. 2 Issue 8, August-2013, pp: (30-37), Available online at: www.erpublications.com

Implementation of DCM module for AUTOSAR Version 4.0

Deepika C. K.[1], Bjyu G.[2], Vishnu V. S.[3]
[1]M. Tech. (Embedded Systems), Sree Buddha College of Engineering, Pattoor, Kerala, India
[2]Specialist, Embedded Group, Transportation Business Unit, Tata Elxsi Ltd.; Technopark, Kerala, India
[3]Assistant Professor, Department Of ECE, Sree Buddha College of Engineering, Pattoor, Kerala, India

Abstract: Standardization is very much the trend in the development of automotive electronics. The use of open architectures and configurable components are intended to let developers focus more on the innovative and differentiating aspects of the development process. AUTOSAR (AUTomotive Open System ARchitecture) has the stated goal of standardizing a common open automotive software architecture. It is a standard platform for vehicle software. The components in AUTOSAR are hardware independent. The diagnostic stack in AUTOSAR consists of three modules: DCM, DEM and FIM. The DCM (Diagnostic Communication Manager) implements the diagnostic communication per ISO 14229-1. This paper focuses on the design and development of the DCM module for AUTOSAR 4.0. The Diagnostic Communication Manager (DCM) provides a common API for diagnostic services in the AUTOSAR-Basic Software. DCM checks if the diagnostic services request is supported and if the service may be executed in the current session according to the diagnostic states.

Keywords: AUTOSAR, ECU, DCM, API, CAN.

I. INTRODUCTION

The AUTOSAR consortium was founded to manage the growing electronics complexity and improve cost-efficiency without any compromises with quality as well as reusability. Increasing complexity of automotive embedded software, increasing needs for software reusability and shorter development life cycle all require new software architecture standards. The proposition of AUTOSAR attempts to provide solutions to these problems. Also AUTOSAR, open and standardized automotive software architecture, is widespread in automotive industry worldwide. In AUTOSAR, the application software and infrastructural software are clearly separated through the concept of RTE (Run Time Environment), and the flow of design is shifted from coding to configuration. The Electronic Control Unit (ECU) software has been already implemented by many OEMs (Original Equipment Manufacturers) in Luxury cars. But the software is not standardized. Many systems used their own protocols which meant that garages had to have a large number of tools – even to diagnose a single vehicle. To overcome this, standards were agreed–starting at the physical level. AUTOSAR (AUTomotive Open System ARchitecture) is one such standard for ECU Software. AUTOSAR is open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers. The major objectives of AUTOSAR are standardization of basic systems functions, scalability to different vehicle and platform variants, transferability throughout the network, integration from multiple suppliers, maintainability throughout the entire product life cycle, software updates and upgrades over the vehicle's lifetime.

II. SOFTWARE ARCHITECTURE

To make a component based design possible, the AUTOSAR uses a layered architecture that ensures the decoupling of the functionality from the supporting hardware and software services. The AUTOSAR Architecture shown in Fig. 1 distinguishes on the highest abstraction level between three software layers: Application layer, Runtime Environment and Basic Software layer which run on a Microcontroller. The uppermost layer is the application layer; which contains all the application specific hardware components. The diagnostics allows the automotive technician to interact with the vehicle's on-board controllers, gathering information, controlling the vehicle directly and aiding in the diagnosis of problems. This interaction can occur with the use of a scan tool or in some case accessing the on-board controller directly through an operation sequence or a control panel on the vehicle.

The AUTOSAR Basic Software is further divided in the layers: Services, ECU Abstraction, Microcontroller Abstraction and Complex Drivers The Basic Software Layers are further divided into functional groups. Examples of Services are System, Memory and Communication Services. The Microcontroller Abstraction Layer is the lowest software layer of the Basic Software. It contains internal drivers, which are software modules with direct access to the microcontroller and internal peripherals. Its task is to make higher software layers independent of microcontroller. The ECU Abstraction Layer interfaces the drivers of the Microcontroller Abstraction Layer. It also contains drivers for external devices. It offers an API for access to peripherals and devices regardless of their location (μC internal/external) and their connection to the microcontroller (port pins, type of interface). Its task is to make higher software layers independent of ECU hardware layout. The Complex Drivers Layer spans from the hardware to the RTE. Its task is to provide the possibility to integrate special purpose functionality. Service layer is present within Basic Software and it provides basic services to the application layer, the basic services includes memory services, diagnostic services, communication services, system services. The AUTOSAR Software Components communicate with other components (inter and/or intra ECU) and/or services via the RTE. The task of this layer is to make AUTOSAR Software Components independent from the mapping to a specific ECU.

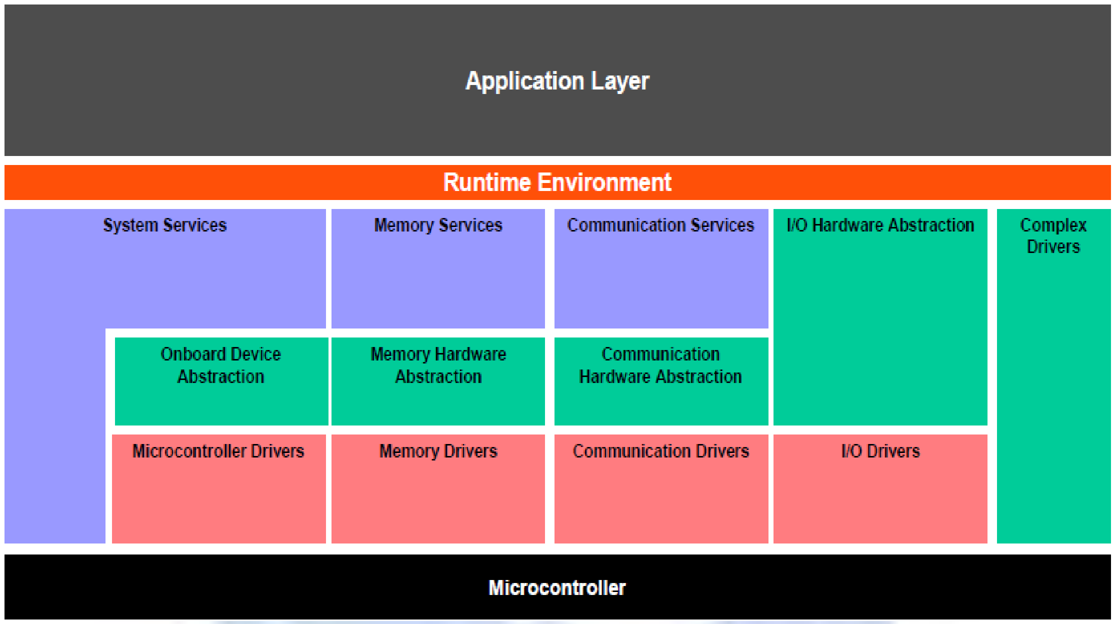

Fig: 1 Detailed architecture

III. DIAGNOSTIC COMMUNICATION MANAGER

The DCM module provides a common API for diagnostic services. The functionality of the DCM module is used by external diagnostic tools during the development, manufacturing or service. The DCM module ensures diagnostic data flow and manages the diagnostic states, especially diagnostic sessions and security states. Furthermore, the DCM module checks if the diagnostic service request is supported and if the service may be executed in the current session according to the diagnostic states.

The DCM handles different diagnostic protocols. One protocol is used for legislated OBD (ISO 15031-5). Another protocol is used for enhanced diagnostics (ISO 14229-1). The DCM provides for all diagnostic services (full-set of ISO 14229-1 and ISO 15031-5) interfaces to the AUTOSAR-RTE. In the AUTOSAR Architecture the DCM is located in the Communication Services (Service Layer).

In the Communication process the DCM receives a Diagnostic message from the PDU Router. In DCM internally the Diagnostic message will be processed, checked and handed on to the AUTOSAR SW Components for further processing. Depending on the Diagnostic service Id the corresponding calls in the Application Layer will be done. The DCM shall be network independent. This requires a network independent interface to the PDU Router (which handles the networks CAN, LIN, FlexRay).

A. Components of DCM

DCM module as consisting of the following submodules:

- Diagnostic Session Layer (DSL) Submodule: The DSL submodule ensures data flow concerning diagnostic requests and responses, supervises and guarantees diagnostic protocol timing and manages diagnostic states (especially diagnostic session and security).
- Diagnostic Service Dispatcher (DSD) Submodule: The DSD submodule processes a stream of diagnostic data. The submodule receives a new diagnostic request over a network and forwards it to a data processor. Transmits a diagnostic response over a network when triggered by the data processor (e.g. by the DSP submodule). The DSD submodule shall only process valid requests and shall reject invalid ones.
- Diagnostic Service Processing (DSP) submodule: The DSP submodule handles the actual diagnostic service (respectively subservice) requests. When receiving a function call from the DSD submodule requiring the DSP submodule to process a diagnostic service request, the DSP always carries out following basic process steps.

  1. Analyze the received request message,
  2. Check format and whether the addressed subfunction is supported,
  3. Acquire data or execute the required function call on the DEM, SW-Cs
  4. Assemble the response Equations

B. Diagnostic Session

Diagnostic session is the basis for communication between the ECU and the diagnostic tool. During ‘Diagnostics’ the ECU being analyzed is in a particular session. Basically there are different types of diagnostics sessions :

1. Default Session
2. Extended Diagnostic Session
3. ECU Programming Session

After Ignition on, ECU will be switched to a Default Diagnostic Session and after receiving the request from Diagnostic Tool, the ECU will be switched to the Extended Diagnostic Session. Further, after receiving the ECU Programming Session start request from Diagnostic tool, it will switch to the ECU Programming Session.

IV. DESIGN

The Diagnostic Communication Manager (DCM) provides a common API for diagnostic services in the AUTOSAR-Basic-SW. The DCM receives a Diagnostic message from the PDU Router. DCM internally the Diagnostic message will be processed, checked and handed on to the Autosar SW Components for further processing. Depending on the Diagnostic service Id the corresponding calls in the Application Layer will be done. Design of DCM module consists, of both High Level Design (HLD) and Low Level Design (LLD).HLD specifies, the identification of APIs needed for the implementation of DCM and the LLD point out the detailed design of APIs. High level and low level design was done by using Enterprise Architect version 9.3 software.

A. High Level Design of DCM

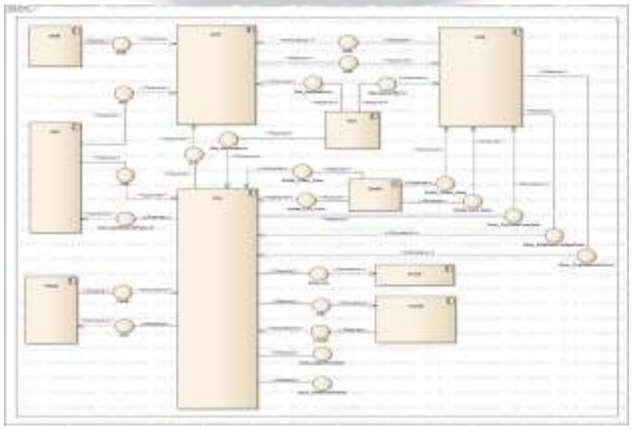

Fig: 2 High Level Design of DCM

The purpose of the DSD (Diagnostic Service Dispatcher) is to process a stream of diagnostic data. Receive a new diagnostic request over a network and forward to a data processor. Transmit a diagnostic response over a network when triggered by the data processor. (e.g.: DSP). The Diagnostic Session Layer (DSL) ensures data flow concerning diagnostic requests and responses. DSL also supervises and guarantees diagnostic protocol timing. Furthermore DSL manages diagnostic states (esp. diagnostic session and security). The Diagnostic Session Layer (DSL) ensures data flow concerning diagnostic requests and responses.

DSL also supervises and guarantees diagnostic protocol timing. Furthermore DSL manages diagnostic states (esp. diagnostic session and security) and communications states required by the Communication Manager. The DSP provides an interface to the DSD and implements various diagnostic services as defined for different protocols like UDS or OBD. DSP - Process specific Service (respectively Sub Service) Requests .The DSP is mainly a container for completely implemented diagnostic services that are common amongst the different applications (e.g. access to the fault data) and thus do not need to be implemented by the application.

B. Low Level Design of DCM

Low level design is done by drawing the activity diagram/ flow chart of APIs realized by Diagnostic Communication Manager. It defines the internal logic of the APIs. The flow chart for some of the APIs that are realized are given below:

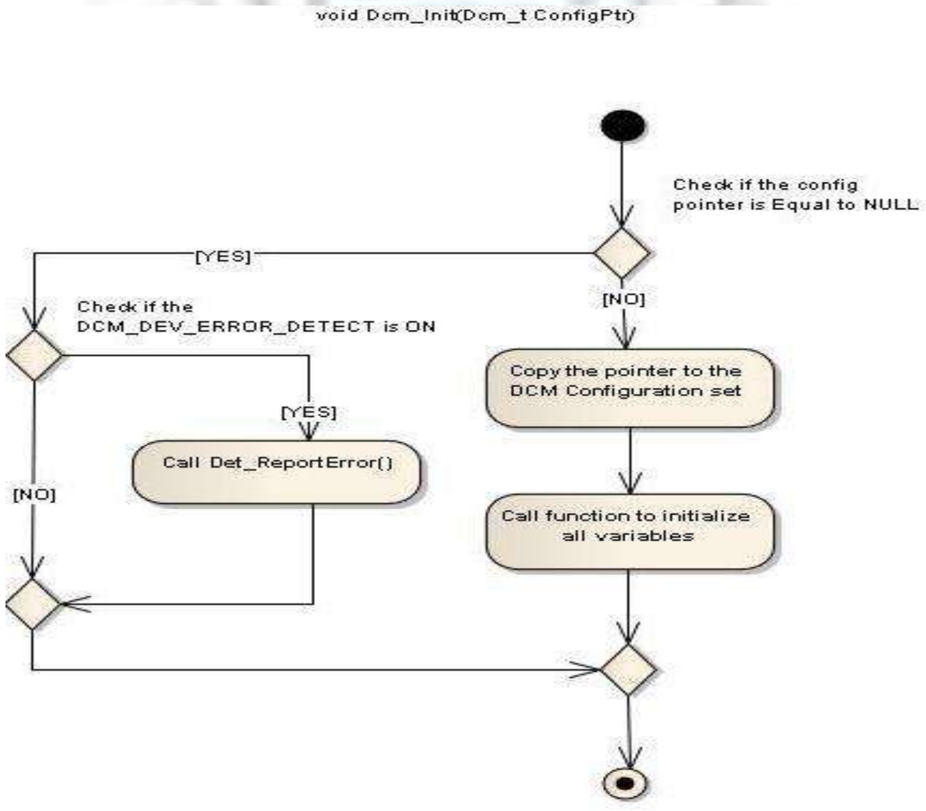
void Dcm_Init(Dcm_t ConfigPtr)
Check if the config pointer is Equal to NULL
[YES]
[NO]
Check if the DCM_DEV_ERROR_DETECT is ON
[YES]
Call Det_ReportError()
[NO]
Copy the pointer to the DCM Configuration set
Call function to initialize all variables

Fig: 3 Flow Chart of Dcm_Init()

V. CONFIGURATION

The configuration of DCM module is done by using the configuration tool eZyConfig. The routing table is configured during the post build time and the parameters corresponding to minimum routing is configured at link time. This tool is developed by Tata Elxsi Limited. In order to build AUTOSAR-compliant software for an ECU, the developer has to depend on configuration tools available in market, since manual configuration is time consuming. Moreover each OEM would be having a specific requirement that needs to be achieved. These requirements can be achieved by extending the standard AUTOSAR specification like adding vendor specific modules, containers, parameters etc.

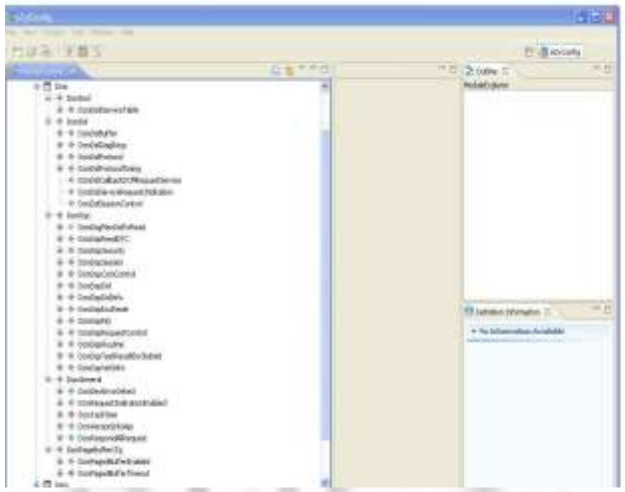

Fig. 4: Configuration Window in eZyConfig

VI. IMPLEMENTATION AND TESTING OF DCM MODULE

The development of DCM is done by coding all the APIs realized by it. Coding is done using 'C' Language. The file structure for the development consists of header files and code files (source files).

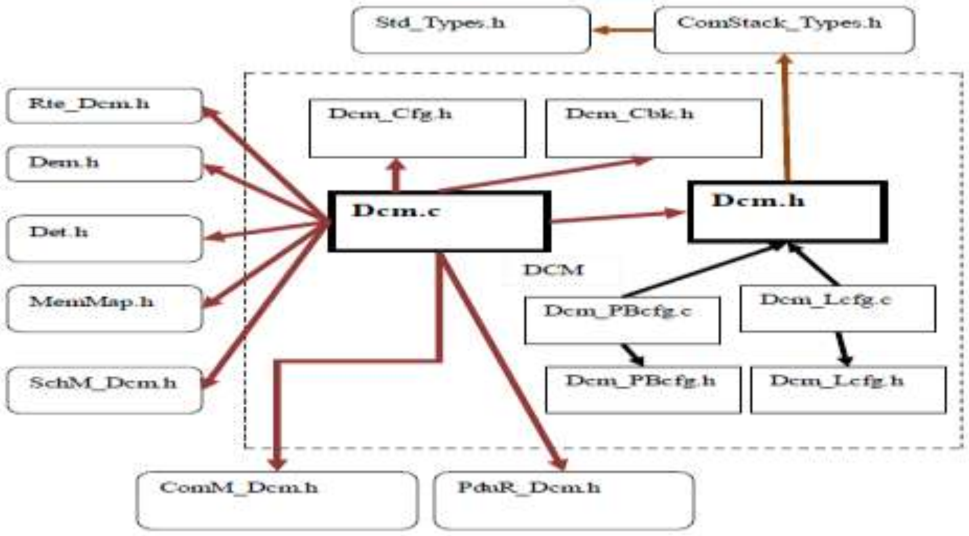

Fig. 5: DCM module file structure

Dcm.h includes general Diagnostic Communication Man-ager(DCM) definitions. Type definitions of the DCM are defined in Dcm.h. Pre-compile-time configuration data of the DCM module is defined in Dcm_Cfg.h. Dcm_<module>.h includes the declaration of APIs of DCM used by different mod-ules. <module> or <module>_cbk .h declares the datatypes and APIs of other mod-ules used by DCM module. Det.h, Dem.h and Dem_IntErrld.h declares the datatypes and APIs of DET and DEM used by DCM module. ComStack_Types.h contains all types that are used by different modules of the communication stack of the basic software that are platform and compiler independent and Std_Types.h contains all types that are used by different modules of the basic software that are platform and compiler independent. Compiler.h and Platform_Types.h declares all the types used by different modules that are compiler and platform dependent. Dcm.c defines all the APIs realized by DCM module and it's header file structure is given in Fig.5

The coding was done in C by using Visual C++ 2008 express edition. It is successfully compiled and build without any error. Figure 6 shows the snap shot of code build in Visual C++ 2008 express edition.

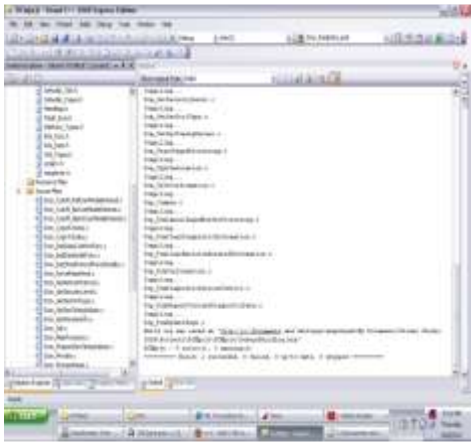

Fig. 6: Snap shot of code build in Visual C++ 2008 express edition.

Testing is an activity in which a system or program is executed under specified conditions, the results are observed, recorded and an evaluation is made out of it.
Testing or validation of DCM module is done in two ways. They are:

1. Unit testing
2. Integration testing

Unit testing is a software verification and validation method in which programmer tests if the individual units of source code are fit for use. The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. Most control flow bugs can be easily detected in unit testing. Test Systems 2.3 (TESSY) is used for Unit testing of DCM module. Tessy performs automatic dynamic unit testing of embedded software and facilitates all other aspects of software testing. Tessy may accept deviations from the exact results. A test report will then be generated in configurable levels of detail and in various formats such as html, doc, xls, or txt. Integration testing mainly deals with testing for the correctness of the interface. It verifies all data exchanged across the inter modular interfaces agree with the pre defined requirement specifications. It also involves the verification of basic functionality of the BSW modules after integration. Validation platform used for integration testing is MPC 5668G Evaluation board.

VII. RESULTS

All the APIs (Application Programming Interfaces) of DCM module is developed. The High level Design (HLD) and Low level Design (LLD) is done using the tool Enterprise Architect version 9.3. The C code is written and compiled in Visual C++. The unit testing of module is done in TESSY 2.3. In unit testing, each function is compiled successfully using GNU Cross Compiler (GCC).

A. Unit Testing Result

Test Systems 2.3 (TESSY) is used for Unit testing of DCM module. Tessy performs automatic dynamic unit testing of embedded software and facilitates all other aspects of software testing. Tessy starts off by analyzing the source module and then lets the user specify the function to be tested. It then identifies parameters of this function, such as inputs, outputs or both and any external or global variables associated with the function and the subroutines within it. Initial results are displayed in the Test Interface Editor (TIE), where they can be modified. Using the Test Definition Editor (TDE), test cases for functions are then defined by specifying values for input parameters, the expected results and how to compare the actual results with the expected results to determine if a test case has passed or failed. The test cases are saved automatically in a database. Tessy is able to export and import test data to and from other tools e.g. Excel. This includes data from CTE (Classification Tree Editor) test case specifications. The Classification Tree Editor shown in Fig. 7 is a graphical tool that supports the Classification Tree Method. In the upper window, the classification tree is drawn. In the lower window, a line in the combination table specifies a test case. You can annotate information such as state descriptions, expected results, values for classes to the tree as well as to the test case specifications. The editor is able to export test case specifications to Tessy and in various file formats such as plain text or HTML and to Tessy.

The maximum and minimum values should be given in these variables to test the boundary conditions and some intermediate values for the test coverage of the function. The expected output of each function must be described in this CTE diagram.

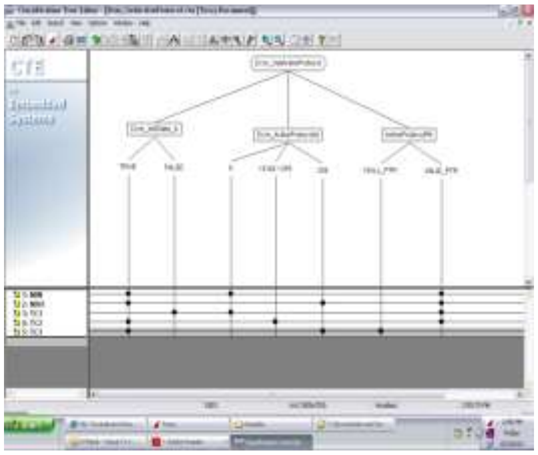

Fig 7: Snapshot of Classification Tree Editor (CTE)

Next step is to give the path of the source file and header files and also select the compiler in the properties of TESSY project file. Each function is compiled in unit testing using GNU Cross Compiler (GCC). The interfaces should be specified by mentioning the external variables and function parameter variables as in, out or inout in Tessy Interface Editor (TIE) as shown in Fig 8.

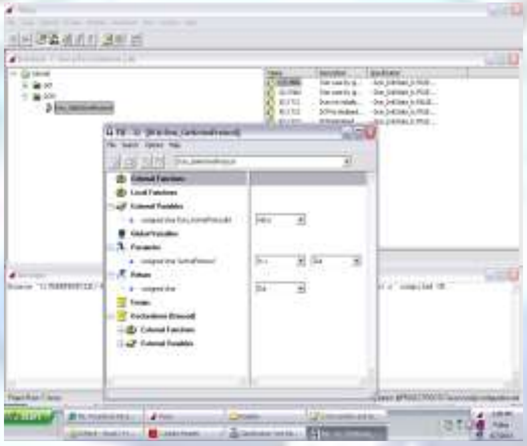

Fig 8: Snapshot of Tessy Interface Editor (TIE)

The possible value for each parameter is given in Test Data Editor (TDE) for each and every test case. The call trace that is the other function called by this function and the number of times it is called has to be given in the expected call trace. The final test report is shown in Fig 9.

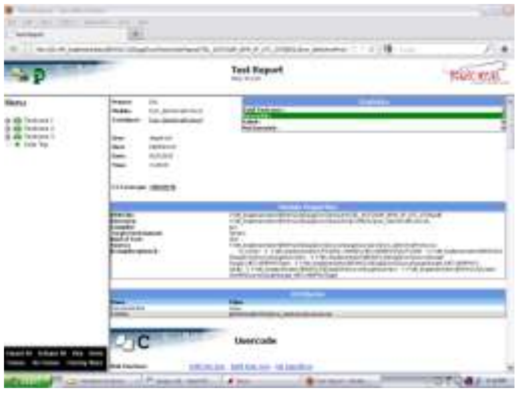

Fig 9: Snapshot of Test Report

B. Integration Testing in Evaluation Board

Integration testing mainly deals with testing for the correctness of the interface. It verifies all data exchanged across the inter modular interfaces agree with the pre defined requirement specifications. It also involves the verification of basic functionality of the BSW modules after integration. Validation platform used for integration testing is MPC 5668G Evaluation board. Fig 10 shows the pictorial representation of MPC 5668G evaluation board.

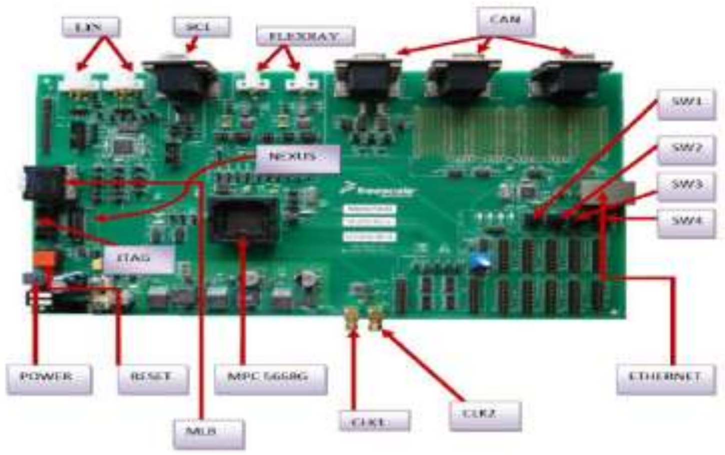

Fig 10: MPC 5668G Evaluation Board

Source files of all the modules used for integration testing are build by using the compiler Diab 5.7.0.0 and Lauterbach as debugger. Diab 5.7.0.0 will support the Processor Architectures like PowerPC, ARM, ColdFire, MIPS, MCORE, Intel, SPARC etc. Its key features include powerful optimization technology, the latest industry standards, one compiler- many architectures, reliable quality etc. The executable is generated in .elf format. The generated executable is flashed to the target board MPC5668G. Vector CANoe is the tool used for integration testing.

VIII. CONCLUSION

Due to much technological advancement, today's cars are more complex than ever before and so the troubleshooting of car's problem has even become more complex and tedious job. The reason for the selection of Autosar as the standard is that Autosar defines a diagnostic platform and it seems all major OEMs are moving towards Autosar. It is important to consider Autosar incase OEMs only hire third party developers in the future who are prepared for Autosar. For the implementation of Autosar based DCM , first the requirements are analyzed, high level design and low level designs are prepared in Enterprise Architect version 9.3 software. Based on these analyzed requirements an algorithm for each API is designed. These drivers are tested in unit level and module levels using TESSY 2.3. The functionality is verified by creating an experimental setup. After the verification the modules are integrated using MPC 5668G to get the overall diagnostic and communication results.

REFERENCES

[1]. AUTOSAR, Specification of Diagnostic Communication Manager V4.2.0 R4.0 Rev 3, 2011. [Online]. Available.http://www.autosar.org/, 2011.
[2]. AUTOSAR, Release 4.0 Overview and Revision History V1.2.1 Release 4.0 Rev 3, 2012. [Online]. Available.http://www.autosar.org/, 2012.
[3]. AUTOSAR Partnership, "AUTOSAR_Technical OverviewV4.2.0R4.0Rev3". [Online]. Available.http://www.autosar.org/download/R4.0/AUTOSAR_TechnicalOverview.pdf 2011.
[4]. AUTOSAR, Technical Overview, V1.2.1, R4.0, Rev3. [Online] Available.http://www.autosar.org/, 2008.
[5]. Diagnostics with AUTOSAR and ODX – Part 1: Diagnostics with AUTOSAR, Technical Article.
[6]. eZyConfig manual,Tata Elxsi Limited,November, 2009.
[7]. "Road Vehicles-Unified Diagnostic Services(UDS): Part 1 Specifications and Requirements", ISO/DIS 14229 - 1.2.
[8]. AUTOSAR, Specification of Diagnostic Communication Manager V2.0.1, 2006 [Online] Available.http://www.autosar.org/, 2008.
[9]. AUTOSAR, Specification of Diagnostic Communication Manager V3.0.0 R3.0 Rev 0001, 2007. [Online] Available.http://www.autosar.org/, 2007.
[10]. MPC5668G Microcontroller, Reference Manual Doc. No. MPC5668XRM Rev.2, Freescale Semiconductor, September, 2008.