International Journal of Technical Research and Applications e-ISSN: 2320-8163,
www.ijtra.com Volume 3, Issue4 (July-August 2015), PP. 230-235

ECU State Manager Module Development and Design for Automotive Platform Software Based on AUTOSAR 4.0

Dhanamjayan P.R.[1], Kuruvilla Jose[2], Manjusree S.[3]
[1]PG Scholar, Embedded Systems, [2]Specialist, Automotive Electronics,
[3]Assistant Professor. Department of Electronics
Sree Buddha College of Engineering, Kerala, India,
[2]Transportation Business Unit Tata Elxsi Ltd,
Kerala, India
[1]dhanamjayan.pr@gmail.com
[3]manjusree.s1@gmail.com

***Abstract*— — As the number of vehicles is increasing day by day, the infrastructure requirements associated with vehicles are also become complex. Hence the count of ECU used in vehicles, in order to satisfy these requirements are growing. As a result complexity increases. The complexity of ECU can be reduced to a great extend by using a standardized architecture. Hence AUTOSAR (AUTomotive Open System Architecture) came into existence and got much popularity in the automotive domain. AUTOSAR is an open and standardized platform for automotive software. By using AUTOSAR standardization, scalability, increased quality and safety of E/E systems can be achieved. EcuM (ECU State Manager) implements the Ecu state management in AUTOSAR platform. EcuM is module present in system services layer of AUTOSAR. EcuM is responsible for initialization and de-initialization of OS and other basic software modules. It also manages all the wakeup events associated with ECU. This paper focuses on the design and development of EcuM module based on AUTOSAR 4.0.**

Index Terms **:API, AUTOSAR, ECU, EcuM.**

I. INTRODUCTION

The AUTOSAR is an open and standardized layered automotive software architecture jointly developed by automobile manufacturers, suppliers and tool developers. The scope of AUTOSAR includes all the vehicle domains. For attaining software reusability with shorter development life cycle requires new software architecture. AUTOSAR is used as a standardized infrastructure for automotive application software development. Different ECU manufacturers uses their own software for ECUs . Hence the software associated with it is not standardized. In vehicles a large number of ECU are used for implementing different applications. If the ECU software is not standardized it becomes almost impossible to integrate the ECU to complex vehicle network infrastructure. Then it becomes costlier and complex. Inorder to overcome these, AUTOSAR architecture is widely accepted in automotive domains. It gives increased flexibility and scalability to transfer and integrate functions, cost optimization of scalable systems, flexibility for product modification and updates, improved reliability and quality of E/E systems.

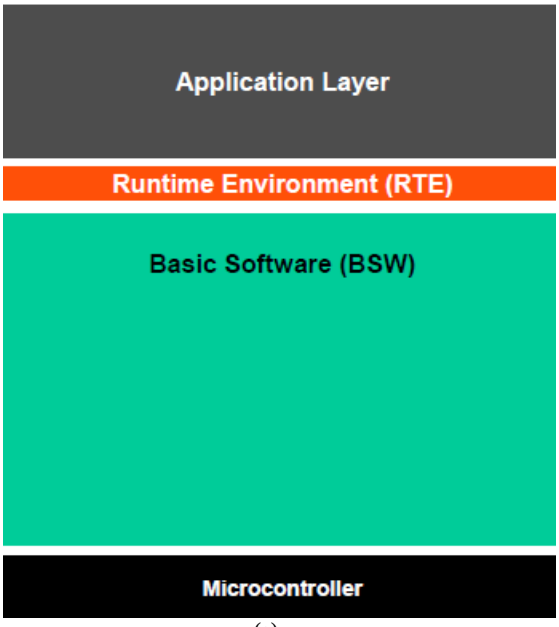
The rest of the paper is organized as follows. Software architecture is explained in section II. The different phases of ECU state manager is discussed in Section III. Section IV describes about the high and low design. The configuration is explained in Section V. the implementation and testing methods are illustrated in Section VI. Experimental results are presented in section VII. Concluding remarks are given in section VIII.

II. SOFTWARE ARCHITECTURE

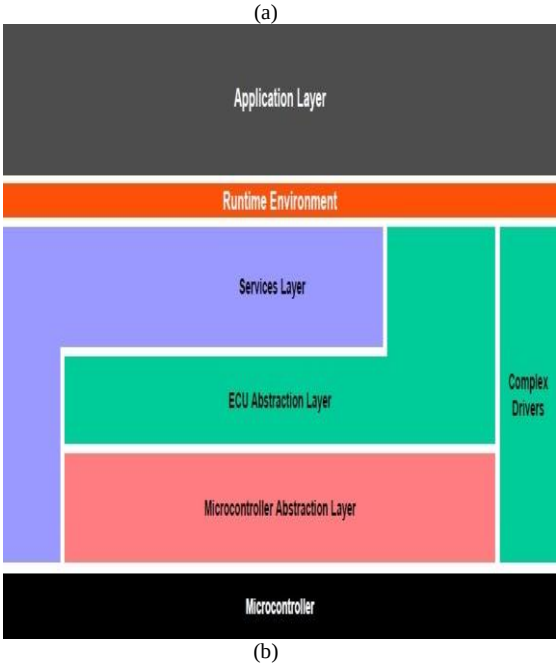
AUTOSAR layered architecture describes hierarchical structure of software with relation and mapping of software layers with basic software modules. This architecture gives strong interaction between application software and underlying hardware components like sensors, actuators and microcontroller hardware. Fig 1(a) shows the AUTOSAR architecture that distinguishes on the highest abstraction layer between three software layers. The three software layers are Application layer, Runtime Environment (RTE) and Basic Software which runs on a microcontroller. The upper layer is the Application layer which consists of application software components which are linked through an abstract component, named the virtual function bus. The application software components are the smallest pieces that have individual functionality. The middle layer is the RTE, the layer providing communication services to the application software. The lower layer is the Basic Software layer that consists of functional group corresponding to system, memory and communication services.

The Basic Software (BSW) layer in the AUTOSAR is further divided into layers: Services, ECU Abstraction,

Microcontroller Abstraction and Complex Drivers as in Fig.1.(b).



(a)



(b)

Fig 1(a)AUTOSAR layered architecture. (b) AUTOSAR layered architecture with BSW description

The Basic software layers are further divided into functional groups. System services, memory and communication are examples of different services. The lowest software layer of Basic software (BSW) is the Microcontroller Abstraction Layer. It contains internal drivers, which are software modules with direct access to the internal peripherals and microcontroller. It's task is to make higher software layers independent on microcontroller. The drivers of Microcontroller Abstraction Layer are interfaced by the ECU Abstraction Layer. It also contains drivers for external devices. The main task of this layer is to make higher software layers independent of ECU hardware layout. The Complex Drivers Layer spans from the hardware to the RTE. Its main task is to provide the possibility to integrate special purpose functionality like drivers for devices. The Services Layer is the highest layer of the Basic Software which provides basic services for applications and basic software modules. The RTE is a layer providing communication services to the application software. The main task of RTE is to make AUTOSAR software components independent from mapping to a specific ECU. The AUTOSAR Software Components communicate with other components (inter and/or intra ECU) and/or services via the RTE.

III. ECU STATE MANAGER

The ECU State Manager (EcuM) module is a basic software module that manages common aspects of ECU states. EcuM is present in the system services layer of AUTOSAR architecture. Specifically, the ECU Manager module initializes and de-initializes the OS, the BSW Scheduler (SchM) and the Basic Software Mode Manager (BswM) as well as some basic software driver modules. ECU Manager module configures the ECU for SLEEP and SHUTDOWN when requested and manages all wakeup events on the ECU.

The ECU Manager module provides the wakeup validation protocol to distinguish 'real' wakeup events from 'erratic' ones. The ECU Manager module have different phases of operation: STARTUP, UP, SLEEP, SHUTDOWN and OFF phases.

A. STARTUP Phase

The purpose of the STARTUP phase is to initialize the basic software modules to the point where Generic Mode Management facilities are operational. ECU Manager module takes control of the ECU startup procedure. Startup process is done by executing a set of sequence before starting the OS (StartPreOS Sequence) and another set of sequence after starting the OS (StartPostOS Sequence). StartPreOS Sequence includes setting programmable interrupt priorities, initializing BSW modules, checking configuration consistency, setting default shutdown target and then starting the OS. StartPostOS sequence includes initializing BSW Scheduler i.e., Initializing the semaphores for critical sections used by BSW modules and initializing BSW Mode Manager.

B UP Phase

In the UP Phase, the EcuM_MainFunction is executed regularly and its major functions are to check if wakeup sources have woken up and to initiate wakeup validation, if necessary and to update the Alarm Clock timer. Wakeup source are not only handled during wakeup but continuously, in parallel to all other EcuM activities. This functionality runs

in the EcuM_MainFunction fully decoupled from the rest of ECU management.

C SLEEP Phase

The ECU saves energy in the SLEEP phase. Typically, no code is executed but power is still supplied, and if configured accordingly, the ECU is wakeable in this state. The ECU Manager module provides a configurable set of (hardware) sleep modes which typically are a tradeoff between power consumption and time to restart the ECU.
The ECU Manager module wakes the ECU up in response to intended or unintended wakeup events (e.g. EVM spike on CAN line). Since unintended wakeup events should be ignored, the ECU Manager module provides a protocol to validate wakeup events. The protocol specifies a cooperative process between the driver which handles the wakeup source and the ECU Manager

D SHUTDOWN Phase

The SHUTDOWN phase handles the controlled shutdown of basic software modules and finally results in the selected shutdown target OFF or RESET.

E OFF Phase

The ECU enters the OFF state when it is powered down. The ECU may be wakeable in this state but only for wakeup sources with integrated power control. In any case the ECU must be startable (e.g. by reset events).

IV. DESIGN

The ECU Manager Module provides a set of APIs (Application programming Interfaces) that are to be realized for EcuM functionality. Design of EcuM consists of mainly two phases. They are High Level Design (HLD) and Low Level Design (LLD). High Level Design gives the overall system design in terms of functional architecture and the overview of system development. The Low Level Design gives the design of the actual program code which is designed based on High Level design. It defines the internal logic of the corresponding module. HLD and LLD are done by using the design tool Enterprise Architect version 9.3. Enterprise Architect supports a number of methods of modeling business processes using UML as the foundation modeling language.

A High Level Design of EcuM

The High Level Design of EcuM is shown in Fig 2.

B Low Level Design of EcuM

Low level design is done by drawing the activity diagram/ flow chart of APIs realized by Ecu Manager module. It defines the internal logic of the APIs. The flow chart for some of the APIs that are realized is shown in Fig 3.

V. CONFIGURATION

The configuration of EcuM 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 kind of tools are developed by AUTOSAR stack supplier.. In order to build AUTOSAR-compliant software for an ECU, the developer has to depend on configuration tools, 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.

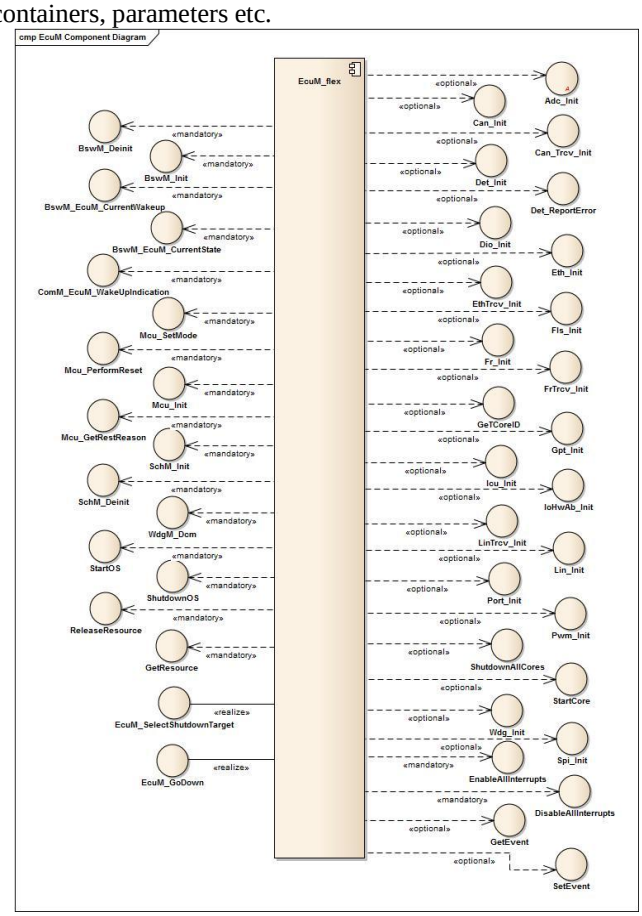


Fig 2. High Level Design of EcuM

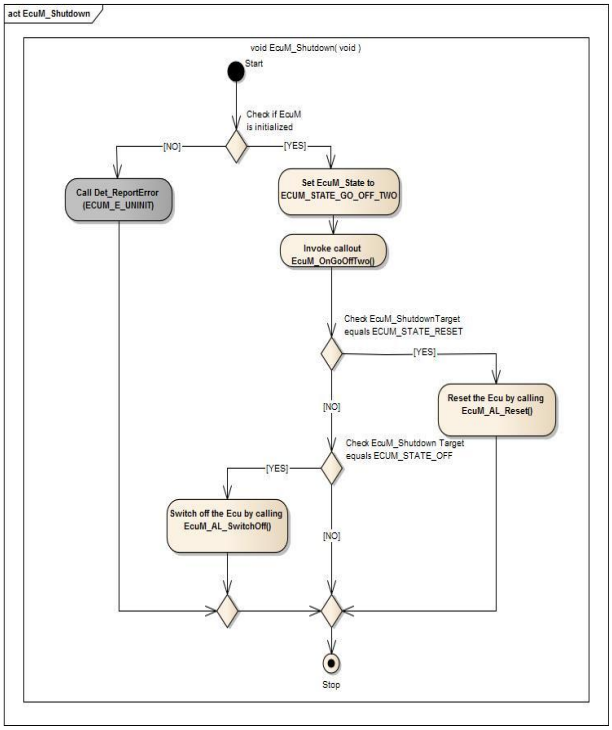


Fig 3 Flow Chart of EcuM_Shutdown()

VI. IMPLLEMENTATION AND TESTING

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).

The ECU Manager module has a well defined code file structure which is shown in Fig. 4. It includes the header file and configuration file required for EcuM. The EcuM module implementation shall provide a file named EcuM.h which contains fix type declarations, forward declarations to generated types, and function prototypes. The ECU Manager module implementation shall provide a EcuM_Generated_Types.h file which contains generated type declarations that fulfill the forward declarations in EcuM.h. It also provides a EcuM_Cfg.h file which contains the configuration parameters. EcuM_Cbk.h file contains the callback/callout function prototypes required for EcuM module. SchM_EcuM.h and MemMap.h are included in ECU Manager module implementation. MemMap.h makes it possible to map the code and the data of the ECU Manager module into specific memory sections. The ECU Manager module shall include the Dem.h file contains the API and Event Id symbol definitions required to report errors. The file EcuM_Types.h shall include Rte_EcuM_Type.h to include the types which are common used by BSW Modules and Software Components. EcuM_Types.h and EcuM.h shall only contain types, that are not already defined in Rte_EcuM_Type.h.

The coding was done in C by using Visual C++ 2010 express edition. It is successfully compiled and build without any error. Fig. 5 shows the snap shot of code build in Visual C++ 2010 express edition.

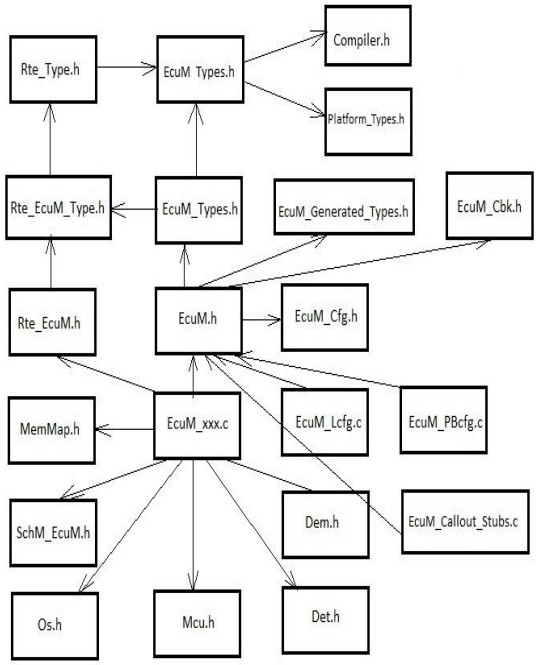


Fig 4. Code File Structure of EcuM Module

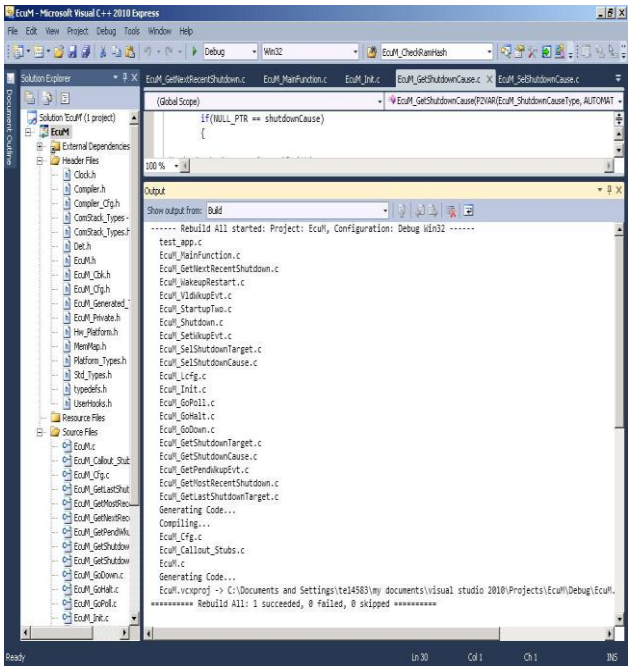


Fig 5 Snap shot of code build in visual C++ 2010 express edition

Testing is done inorder to verify the correctness of module implementation. First module testing is done and then integrated testing has to be carried out. The goal of unit testing is to isolate each part of the program and show that the

individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy. Unit testing finds problems early in the development cycle. This includes both bugs in the programmer's implementation and flaws or missing parts of the specification for the unit.

The unit testing of EcuM can be done by individually validating the API's associated with it. Visual C++ is used for EcuM module testing. Testing application has been written for the module testing.

Once all the individual units are created and tested, integration testing is started by combining those "Unit Tested" modules. The main function or goal of Integration testing is to test the interfaces between the units/modules. The individual modules are first tested in isolation. Once the modules are unit tested, they are integrated one by one, till all the modules are integrated, to check the combinational behavior, and validate whether the requirements are implemented correctly or not. Validation platform used for integration testing is MPC5668G Evaluation board.

VII. RESULTS

The APIs(Application Programming Interface) associated with EcuM module is developed. Using the tool Enterprise Architect version 9.3 High Level Design (HLD) and Low Level Design (LLD) are done. The C code is written in Notepad++ and compiled using Diab compiler. The testing is done with the help of visual C++ 2010 express edition.

A Unit Testing Results

In the unit testing each of the Application Programming Interfaces(API) are tested by written test application C codes. By successfully running the test application API can be validated. After the successful testing of each APIs it can be bulid as a library file. Fig. 6. Shows build output (EcuM.lib) using the diab compiler. The library file is called by all other modules that requires the functionality of EcuM.

B Integration Testing Results

After the creation of EcuM.lib file we have compiled the whole AUTOSAR source by integrating EcuM.lib with other software modules in the AUTOSAR. The successful compilation and building has generated an executable elf file. The screen shot of the AUTOSAR source compilation and elf file generation is shown in Fig. 7

The generated elf file is flashed to MPC5668G Board using Trace32 software. Fig. 8 shows the picture of MPC5668G evaluation board

Through Interactive Generator block, message is send from CANoe to MPC5668G Board and this message can be viewed in Trace window of Vector CANoe. The successful communication between CANoe and MPC5668G evaluation board shows that integrating testing is a success.

Fig 6 Build output of EcuM from diab

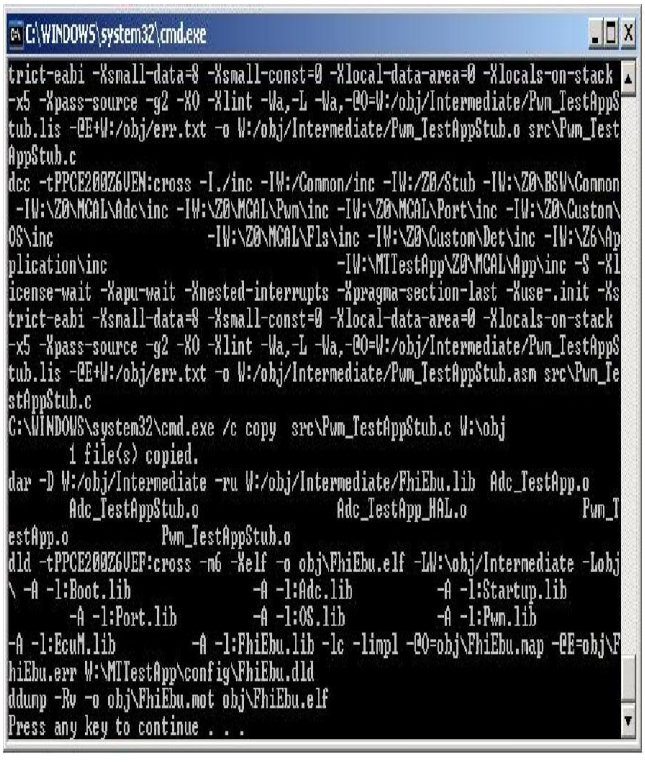


Fig 7 Integrated build output in diab compiler

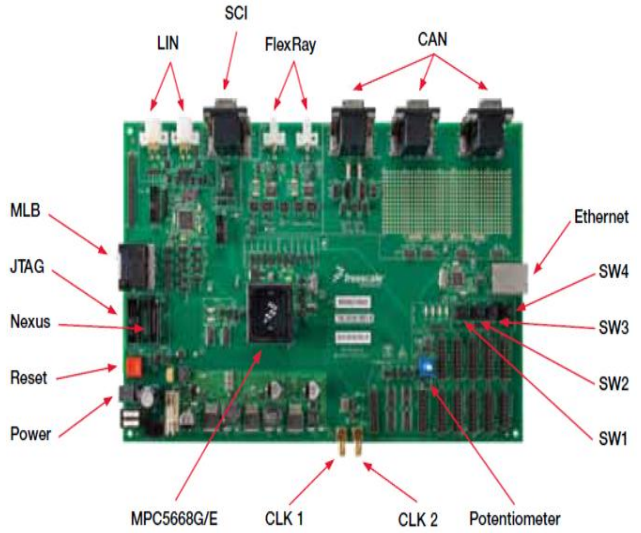


Fig 8 MPC5668G Evaluation board

VIII. CONCLUSION

As the development in automotive industry is in a rapid phase, a standardized architecture is necessary for ECU's in the vehicles to reduce complexity and to increase safety. Thus AUTOSAR architecture is now widely used as a standardized architecture in automotive industry because of its peculiar features. The Ecu State Manager (EcuM) module present inside the services layer of AUTOSAR architecture is being realized. All the APIs (Application Programming Interfaces) of EcuM module is developed. Based on the requirements and specifications the High Level Design and Low Level Design is done using the tool Enterprise Architect version 9.3. The module is then tested using Unit testing and Integrated testing. Unit testing is done by creating sample application in Microsoft Visual C++ for functional validation of APIs. Integration testing is done with the help of MPC5668G evaluation board and result was verified by transmitting and receiving messages between the MPC5668G board and Vector CANoe tool.

REFERENCES

[1] AUTOSAR, Specification ECU State 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] eZyConfig manual,Tata Elxsi Limited,November, 2009
[6] MPC5668G Microcontroller, Reference Manual Doc. No. MPC5668XRM Rev.2, Freescale Semiconductor, September, 2008.