Addis Ababa University

School of Graduate Studies

Faculty of Technology

Model Driven Architecture Approach for Software Development in Embedded System

BY

Gutema Jira

August, 2007

Addis Ababa University
School of Graduate Studies
Faculty of Technology

Model Driven Architecture Approach
for
Software Development in Embedded System

A Thesis submitted to the School of Graduate Studies of Addis Ababa University in partial fulfillment of the requirements for the degree of Master of Science in Computer Engineering.

BY

Gutema Jira

Advisor

Dr. Kumudha Raimond

Addis Ababa

August, 2007

Addis Ababa University
School of Graduate Studies
Faculty of Technology

Model Driven Architecture Approach

for

Software Development in Embedded System

BY

Gutema Jira

Approved by Board of Examiners

Dr. Mengesha Mamo _____

Chairman, Department of Electrical and Computer Engineering Signature

Dr. Kumudha Raimond _____

Advisor Signature

_____

External Examiner Signature

_____

Internal Examiner Signature

ACKNOWLEDGMENTS

I would like to express my thanks to all who helped me to finish this thesis work. Special thanks are owed to my advisor Dr. Kumudha Raimond. She monitored my work and took effort in reading and providing valuable comments on major outcomes of this thesis work.

Similarly, I wish to extend my thanks to Dr.-Ing Hailu Ayele and Dr. Eng. Tamrat Bayle for their valuable comments. I also extend my tanks to my friends Abrham Fikru, Endale Nega, Eyob Alemu, Hassen Numan and others who helped and encouraged me during my thesis work.

I also wish to extend my tanks to the MicroLink Information Technology College for the workload protection and partially financing my thesis and course work. Finally I would like to give tanks to the almighty God for the successful completion of my master program.

Table of Contents

List of Tables

List of Figures

ACRONYMS AND ABBREVIATIONS

API: Application Programming Interface

ASP: Application Specific Programmable

CAN: Controller Area Network

CORBA: Common Object Request Broker Architecture

CRC: Cyclic Redundancy Check

CWM: Common Warehouse Metamodel

DCOM: Distributed Component Object Model

DLC: Data Length Code

EOF: End Of Frame

GRM: General Resource Modeling

I^2C: Inter IC Communication

IDE: IDentifier Extension

IFS: Intermission Frame Space

J2EE: Java 2 Enterprise Edition

LAN: Local Area Network

MDA: Model Driven Architecture

MOF: Meta Object Facility

OMG: Object Management Group

PIM: Platform Independent Model

PM: Platform Model

PSM: Platform Specific Model

QOS: Quality of Service

R/W: Read/Write

RTOS: Real-Time Operating System

RTR: Remote Transmission Request

SOF: Start Of Frame

SPT: Scheduling, Performance, and Time

SRR: Substitute Remote Request

UML: Unified Modeling Language

XMI: XML Metadata Interchange

XML: eXtensible Markup Language

ABSTRACT

Many assumptions change in the life time of embedded system products due to the growing variations in implementation platforms, technological inventions and constantly changing requirements and also embedded systems have limited resources. Concepts such as QoS (Quality of Service) based MDA (Model Driven Architecture) approach solves the above major problems because it enables the software to be developed independent of specific platforms with the awareness of QoS. MDA has a means for system specification independent of the current technology by separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform as two different concerns of development. The two concerns are described as Platform Independent Model (PIM) and Platform Specific Model (PSM).

This thesis work focuses on the priority based embedded networks such as I^2C and CAN, which employ a connectionless mode services for QoS aware applications. The QoS based MDA approach for these networks' services include both the functional and QoS characteristics. The functional characteristics include data transfer and event signaling. The QoS characteristics are based on the frame transmission delay, throughput, error detection delay and priority. A conceptual model that contains PIM, PM, mapping layer, PSM and a possible mapping procedure has been proposed.

Finally, using the I^2C based experimental setup; it is possible to measure the data transit delay and the throughput to compare with the corresponding theoretical values in the PM of the developed conceptual model. Moreover, this experimental setup indicates the possibility of measuring and providing the values of the supported QoS parameters of a given network to the mapping layer of the conceptual model. If the QoS requirements of a given application is provided to the mapping layer, then these requirements will be compared with the values of the supported QoS parameters by the mapping layer in order to produce the PSM that contains the selected appropriate target network for the final implementation.

CHAPTER 1. Introduction

1.1. Overview

The use of computer controlled systems has increased dramatically in our daily life. Embedded processors and microcontrollers are now found in nearly all technical devices: in most of the devices we use everyday such as mobile phones, PDAs, TVs, DVD players and Cameras; in facilities and facility management such as heating, air conditioning, elevators and escalators; in production units from robotics to production automation and control systems; in medicine where equipment for diagnostics and medical support is enhanced and in the increasing variety of intelligent devices that are implemented into the human body. And also the rapid proliferation of embedded systems in transportation such as cars, trucks, ships and airplanes is remarkable [34].

An embedded system is some combination of computer hardware and software, either fixed in capability or programmable, that is specifically designed for a particular kind of application device. The embedded systems are naturally close to the architectural platform (hardware, I/O interconnections, microprocessor and memory) [6]. This causes platform dependency, which requires embedded software developers to spend much time in studying and resolving the platform and technology problems instead of focusing on application requirements and design. For most embedded systems, the development process for both the hardware and software can occur at the same time [21, 32]. This has been the source of complexity and lack of portability in the domain of embedded systems. Embedded systems have also limited resources. Thus the QoS based approach should be applied to efficiently use these resources.

QoS is originated from the domains of distributed systems, networks and embedded systems. QoS is defined in [7] as: “A set of perceivable characteristics in user friendly language with quantifiable parameters that may be subjective (observable) or objective (measurable).” QoS requirements specify how the system satisfies its users while performing its function. The characteristics of QoS and its parameters are expressed on two types of concerns: user satisfaction and resource consumption. User satisfaction is based on the stated user requirements and resource consumption is used by the provider of required service to manage and control the consumed resources.

In a lifetime of embedded system products, many assumptions change - the environment, the hardware platform, communication standards, component models, languages, etc. Old technologies become obsolete [33, 34]. This poses huge problems for maintenance and in design. To improve the application time-to-market; even to make the maintenance possible and to reduce the amounts of time and effort in design and maintenance, a means for system specification independent of the current technology is required. Concepts such as MDA have the goal to allow flexible evolution of the applications and their components. MDA starts with the well-known and long established idea of separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform [37].

1.2. Statement of the problem

Changing existing software due to the growing variations in implementation platforms, technological inventions and constantly changing requirements is becoming almost impossible [7]. Hence a new development approach, MDA, is required. One of the major benefits of this approach is that the application to be developed can retarget a new or different platform just by modifying the mapping layer [3]. And also resource constraints are major issues in embedded systems and hence a QoS based model for efficient utilization of the platform capabilities is needed [2].

As current and future applications are embedded in small communication devices, the network level variability must be handled in a formal development methodology. To make an embedded application design easier and more portable, the application to be developed should be independent of a specific network. This can be achieved by using MDA approach. This work focuses on the communication subsystem particularly the priority based (typically connectionless) networks such as I^2C and CAN. It uses MDA approach to develop a conceptual model for these networks. A specific application that uses the I^2C network is developed to measure some of the QoS parameters such as delay and throughput of this network. Then the measured values are compared with the theoretical values in order to verify the values to be used in conceptual model's PM.

1.3. Review of Related Works

A number of works have been studied and analyzed to be used as a basis for this thesis work. Some of them are presented below.

The work in [10] introduces the notion of abstract platform that can reflect the characteristics of different platforms at a higher abstraction level. An abstract platform defines an acceptable or an ideal platform from an application developer's point of view; it is an abstraction of concrete platform characteristics that can be used in the platform independent models. Alternatively, an abstract platform defines characteristics that must have proper mappings onto the set of concrete target platforms that are considered for an MDA design process, thereby defining the level of platform-independence for this particular process. The domain used to verify the approach for this work is general systems at the enterprise level. The platforms considered are the component-based platforms such CORBA and EJB.

The work in [21] introduces an MDA approach that employs co-design architecture methodology for hardware and the corresponding software. This methodology first creates a PIM for both the hardware and the software, then creates an association PIM by matching the model elements of hardware and software PIMs. Finally, the association PIM is transformed to a deployment PSM that targets specific execution environment. The work does not specify how the QoS issues are dealt with and how the mapping is made. The target platforms considered are buses and chips at the hardware level. It also does not show the Application Programmable Interface (API) level abstractions.

Focusing on the communication subsystem of embedded platforms, the work in [7] introduces an MDA approach for the development of embeddable communicable applications. A QoS aware and resource oriented approach between applications and platforms were proposed. Reservation based (typically connection oriented) networks such as Bluetooth and IrDA are considered. The conceptual model and the mapping strategy for these networks as target platforms have been defined. However, the priority based (typically the connectionless) networks such as I^2C and CAN are not considered in this work.

In [26], it is stated that modern automotive applications become more and more complex: they are implemented over distributed architectures that include several electronic control units

(ECUs) communicating via a local communication network which is event triggered (e.g. CAN, controller area network) or time triggered (e.g. TTCAN, time triggered CAN). A key design issue of such embedded systems is to satisfy a number of QoS constraints such as real-time, dimension, fault tolerance, etc. This paper presented a formal evaluation of the communication protocols used in automotive applications (with CAN/TTCAN). The evaluation is done using three criteria: throughput (dimension constraint), error detection delay (fault tolerance constraint) and information transmission delay (real-time constraint). The work identified, for each communication protocol, different classes of application to help automotive system designers to choose the best protocol that fits the applications requirements. However, the work does not consider the QoS based MDA approach in order to select the appropriate target network so that there will be flexible reusability of the applications on to different or new platforms.

The above works have shown various aspects of modeling at different levels of abstractions and domains. Moreover, many of the recent works largely concentrate on extending the modeling languages typically the UML towards the domain of embedded systems and QoS. However, Mapping QoS aspects at different abstraction levels is the least considered area, which is a crucial aspect for the adaptation of MDA towards the embedded systems domain.
The works lack a complete MDA approach that can enhance applications development in the domain of embedded systems. And also no work has been done that has considered the priority based networks that are very essential for real time applications in the domain of embedded systems with the MDA approach. The SoC (System on Chips) domain [21], the work in [7] and others have shown a proof of the applicability of MDA approach in the embedded systems domain.

Based on the related works and background study, a QoS aware MDA approach for software development in embedded system is proposed in this work. Focusing on the network subsystem of embedded platforms particularly on the priority based embedded networks such as CAN and I^2C, a QoS aware MDA approach that will help the development of embedded applications independent of a specific network is proposed. An I^2C based experimental setup will be implemented in order to measure the values of the supported QoS parameters such as delay and throughput of an I^2C network at different bus lengths.

1.4. Research Objective

The objective of this research is to propose a conceptual model using an MDA approach for the priority based embedded networks, which includes a generic model (PIM) and a mapping methodology towards a specific network model (PSM). In order to measure some of the supported QoS parameters for I^2C network that are used in the PM of the conceptual model (refer to figure 4.4), a specific application is developed using I^2C as the target platform.

The general nature of embedded applications is their QoS requirement, which comes from the need to effectively utilize the limited resources of the target platforms. This work has focused on MDA approach that can support model based QoS aware development of future embedded application on devices with priority based embedded networks.

Specific objectives

- Analysis of the connectionless embedded networks: The major services and QOS parameters of CAN and I^2C will be defined.
- Modeling the services and the QOS parameters of I^2C and CAN: Elements of the PM that should be included will be proposed.
- Modeling the platform independent services and QOS specifications of a generic network model (PIM).
- Proposing the mapping procedure to transform the PIM to a PSM. The mapping procedure specifies the QoS aspects for the transformation.
- In order to measure some of the supported QoS parameters for I^2C network that are identified in the PM of the conceptual model, a specific application is developed using I^2C as the target platform. Then the measured values are compared with the theoretical values in order to verify the values of the supported QoS parameters that will be specified in the conceptual model's PM.

1.5. Methodology

The following is the methodology followed to accomplish the objectives stated above.

►Background Study

The embedded systems and the MDA approach have been studied in order to understand the rest of the document and the thesis work.

►Analysis of I^2C and CAN networks

The major services and QOS provision characteristics and parameters of these networks are defined.

►Model Development

After the modeling elements are determined, the networks are described according to the UML (Unified Modeling Language) concepts. The modeling elements for PM are defined to characterize these networks. Next, the corresponding PIM that can appropriately abstract the services and QOS parameters of the platform are defined. Then, the appropriate mapping strategy that can transform services and the QOS characteristics of the PIM and the PM to produce the PSM is proposed.

►Implementation

I^2C based experimental setup will be implemented to measure the data transit delay and the throughput. The measured values will be compared with the corresponding theoretical values in the PM of the conceptual model.

1.6. Outlines of the thesis

The rest of the thesis is organized as follows:

In chapter 2, the basic concepts that are fundamental for understanding the rest of the document will be introduced. First, it will discuss about the basic characteristics and challenges in embedded systems and application development in this domain. It will also discuss about MDA and its concepts, standards and success in software development. Finally, it discusses the QoS and the UML profiles that are used as a basis for defining the models in the conceptual model of this work.

In chapter 3, an overview of priority based networks such as CAN and I^2C will be given.

In chapter 4, the overview of the conceptual model in this thesis work will be presented. It then continues with the detailed analysis of the QoS parameters of I^2C and CAN and the modeling of the platforms focusing on QoS. The proposed PIM and mapping procedure are also incorporated. Finally, the proposed conceptual model that includes the PIM, PM, Mapping, PSM and logical relationships between them is presented.

In chapter 5, the implementation that uses the I^2C as target network will be presented.

In chapter 6, major contributions of this thesis work will be presented; conclusions of the entire work and also the results from the experiments are concluded.

In chapter 7, possible future works and limitations will be presented.

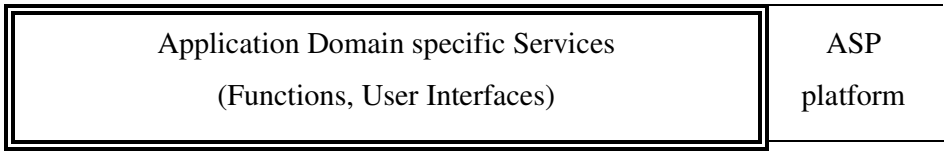
CHAPTER 2. General Background

Embedded Systems

The early embedded systems have specific purposes, very limited storage capacity, relatively slow processing speeds, and small data bus sizes. Technological advances in hardware brought about an evolution in embedded systems. These technological advances enabled the embedded systems to have improved storage capacity, processing speeds, data bus sizes and extensive interfaces. These result in incorporating the processing and communication functions in modern embedded systems. This fact has made the embedded systems to host applications, to process and interchange information, to share resources. As a result, embedded system is enhanced from specific to more general functionality and of more intelligent capability. Further, new development approaches for embedded software such as programmable interfaces and software abstraction layers support flexible system development. This increases the possibility of designing the applications independent of a specific target platform [11, 4].

Embedded System Platforms

Platform is in general defined by Object Management Group (OMG) [15] as "A platform is a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented." Every platform gives a perspective of mapping of higher abstraction layer to it and defines the class of lower level abstraction that it implies. For embedded system, the platform is a fixed micro architecture that masks details but flexible enough to work for a set of applications so that production volume remains high over an extended platform life time. With platform based design methodology, the embedded systems' platforms are modeled at different abstraction levels so that developers could choose the appropriate abstraction level that can avoid their concern about details of the Platforms [1]. A typical architecture of an embedded platform is shown below in figure 2.1.



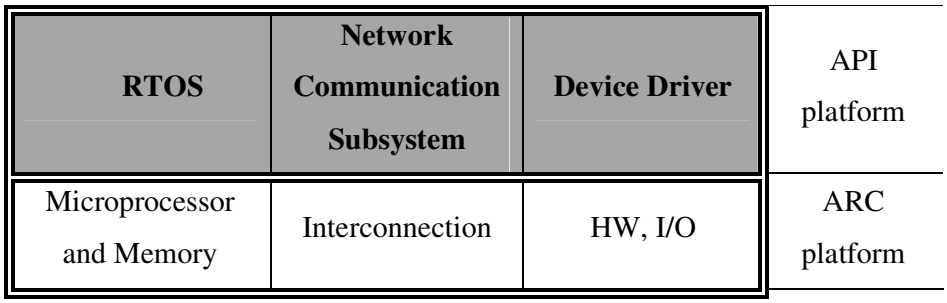


Figure 2.1 **Platforms at different levels**

As shown in figure 2.1, there are three abstraction levels: ARChitecture (ARC), Application Programming Interface (API), and Application Specific Programmable (ASP) Platforms. The ARC Layer includes a specific family of micro-architectures (physical hardware). The API Layer is a software abstraction layer wrapping ARC implementation details. API presents what kinds of logical services are provided and how they are grouped together and represented as interfaces. For example, it may show that data transmission is supported by a communication subsystem, but not how this service is implemented within the hardware. RTOS (Real Time Operating System), device-driver and network communication subsystem are treated as three components of the API platform [1]. ASP is a platform, which provides a group of application domain-specific services directly available to users. In addition to utilizing these existing services, users sometimes also need to modify or combine them, or even develop new services to meet certain requirements. This thesis work focuses on the communication subsystem particularly on the priority based connectionless networks as target platforms for embedded system application development using MDA approach.

Embedded Networks

Generally, the embedded networks are link layer networks and almost all of them are QoS aware. However, each of them has different services with its corresponding interface and different QoS provisions and parameters to define the services. This causes network heterogeneity which is one of the current challenges in software development for embedded systems

To handle this heterogeneous environment and achieve interoperability, some methodology that can abstract the variety in the target network has to be used. Some of the variations of the networks are [7]: Controller Area Network (CAN) and Time Triggered Protocol (TTP) in

automotive systems; Inter IC (I^2C) in ICs interconnections; fire wire and Universal Serial Bus (USB) for multimedia devices and peripherals interconnection; Ethernet and wireless LAN for office automation, industrial automation and multimedia devices and peripheral interconnections; Bluetooth and Infrared Data Association (IrDA) to interconnect peripherals and portable devices.

Model of the Data Link Layer

According to [5], data link service provider is the data link layer entity that provides its services and data link service user is the network layer entity that uses the data link services. As indicated in the figure below, confirmations and indications are messages that come from remote devices and pass through lower layers as events to the upper layers. On the other hand, requests and responses are messages that come from the upper layers to the lower layers.

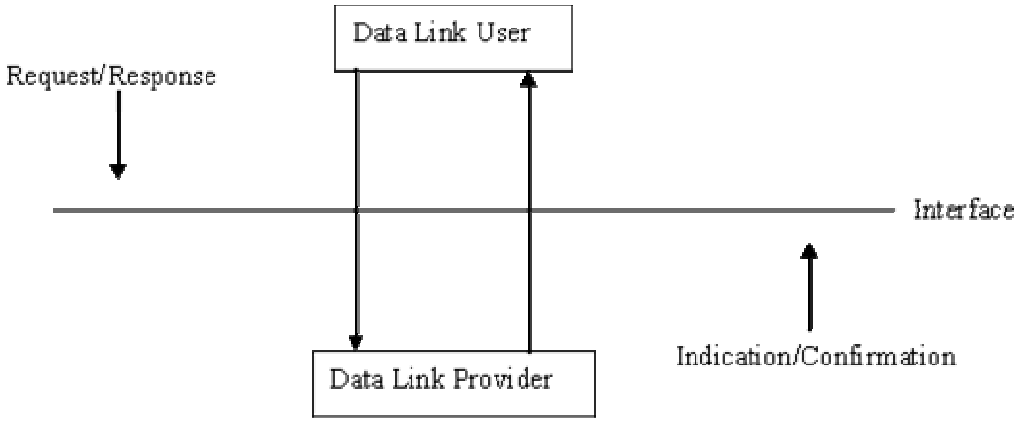


Figure 2.2 **Data Link Services Provider and User**

Broadly, the data link services are classified into two groups. These are: the connection-oriented and connectionless mode of services [5, 29]. The connection-oriented mode is circuit-oriented and enables data to be transferred over a pre-established connection in a sequenced manner. The connectionless mode is message-oriented and supports data transfer in self-contained units with no logical relationship established between units.

Connection-Oriented Mode Service

The connection-oriented mode service is characterized by four phases of communication: local management, connection establishment, data transfer and connection release.

Local management is a phase that enables a user to initialize a stream (communication endpoint) for use in communication and establish an identity with the provider.

Connection establishment is a phase that enables two users to establish a data link connection between them to exchange data. One user (the calling user) initiates the connection establishment procedures, while another user (the called user) waits for incoming connect requests. The called user is identified by an address associated with its stream. A called user may either accept or deny a request for a data link connection. If the request is accepted, a connection is established between the users and they enter the data transfer phase. For both the calling and called users, only one connection may be established per stream. Thus, the stream is the communication endpoint for a data link connection.

Data transfer is a phase, in which the users are considered as peers and may exchange data simultaneously in both directions over an established data link connection. Either user may send data to its peer user at any time. Data sent by a user is guaranteed to be delivered to the remote user in the order in which it was sent.

Connection release is the final phase that enables either the user or the provider to break an established connection. The release procedure is considered abortive, so any data that has not reached the destination user when the connection is released may be discarded by the provider. Examples of connection oriented mode service networks are Bluetooth and IrDA

Connectionless Mode Service

The connectionless mode service does not use the connection establishment and release phases of the connection-mode service. The local management phase is still required to initialize a stream. Once initialized, it enters into connectionless data transfer phase immediately. Though there is no established connection, the connectionless data transfer phase requires the user to identify the destination of each data unit to be transferred. The destination user is identified by the address associated with the data. Connectionless data transfer does not guarantee that data units will be delivered to the destination user in the order in which they were sent. Furthermore, it does not guarantee that a given data unit will reach the destination user, although a given provider may provide assurance that data will not be lost. Examples of connectionless mode service networks are I^2C and CAN.

This thesis work focuses on priority based networks (I^2C and CAN) of the communication subsystem. The main reasons why the I^2C and CAN networks are chosen as the target platforms (in this thesis work) for the embedded applications to be developed using the MDA approach are listed below.

These networks are:

- not addressed as the concrete platforms in PM using MDA approach and they are indicated as the future work in [7].
- very essential and popular in the domain of embedded systems for IC interconnections, real time applications (in automotive for engine control, other parts, in air craft, etc) and others.
- standardized and have well defined protocol and frame formats.

Challenges in Embedded Software Development

Developers in the enterprise system usually develop on the same hardware (host machines) and software platform (Operating System) on which their final product will run. Moreover, developers in the enterprise system use well-tested and well-known development environment that has not changed for a long time. As a consequence of this, software developers focus on the application rather than on the environment.

However, in embedded system, hardware design can be changed to be very different from the architecture of the development host and the last design that the software engineer worked on. Thus, it is often the responsibility of the embedded developer to define and implement the software platform and many of the low level drivers and software stacks; either need to be developed or ported to the hardware platform. Here are some of the difficulties that embedded developer faces: With each different software or hardware platform, there is often a new, different and proprietary development system that is very specific to either the hardware or the software environment; embedded engineer has faced the challenge with the on-going time-to-market issues. Thus, embedded engineers are now looking for technologies that can help them overcome these challenges and become more productive [33, 34].

Additionally, embedded software is different from general software system in the following fundamental ways:

- Object Oriented (OO) techniques such as inheritance, dynamic binding, and polymorphism are rarely used in practice with embedded software development. This is because embedded software is written in low-level language such as Assembly, which lacks the advanced features of OO language;
- Processors used for embedded systems often avoid the memory hierarchy techniques that are used in general purpose processors to deliver large virtual memory space and faster execution using caches;
- QoS is the major concern in the domain of embedded systems in order to utilize the limited resources more efficiently, whereas the middleware platforms such as CORBA, J2EE and DCOM require a lot of resources, give less attention to QoS and hence are not generally used for the domain of embedded systems.
- Automated memory management with allocation, deallocation and garbage collection are largely avoided in embedded software. Further, operating systems with rich suite of services are also avoided.

Thus, the embedded software is much harder to design. In addition to the above mentioned challenges, another cause of the difficulties in embedded software development is that the role of embedded software has changed in the last few decades. Information appliances are rapidly emerging in the consumer electronics market as a result of recent convergence of telecommunication, consumer electronics and information technologies. This poses a great deal of new design challenges to embedded system developers and causes embedded software crisis. Concepts such as QoS aware MDA approach can be used for embedded software development to overcome the above challenges.

Overview of MDA

As the scope and diversity of software systems grow, a number of systems appear to be deployed on different platforms (operating systems, communication protocols, programming languages and hardware) and hence integration between them becomes difficult. As a solution, a middleware that can hide a number of heterogeneous platforms through standard interfaces and high-level communication protocols has emerged. However, recently there are different middleware platforms that can cause heterogeneity at the middleware level. Moreover, from the recent developments, it has been observed that middleware level heterogeneity and evolution will be unavoidable [7].

To address this problem, the OMG was formed to reduce the complexity, minimize the costs, and enhance the introduction of new software applications. The OMG is accomplishing this goal through the introduction of MDA architectural framework with supporting detailed specifications. These specifications will lead the industry towards interoperable, reusable, portable software components and data models based on standard models. The MDA can address the middleware platforms' heterogeneity by separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform. MDA approach is model driven because it uses models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification [3].

The MDA methodology provides platform independence by modeling systems free of the specific artifacts and concepts used in any of the middleware platforms. With this methodology, it is possible to have an architecture that will be language, vendor and middleware neutral. Platform independent applications built using MDA approach may range from small-scale real-time and embedded systems to large-scale distributed enterprise systems. They can be deployed on a range of open and proprietary platforms, such as CORBA, J2EE, .NET [15, 11, 3]. The platform independent applications remain invariant regardless of whether the operation is performed by a CORBA, an Enterprise Java Beans, or a DCOM Object. Thus, a PIM is a formal specification of the structure and function of a system that abstracts away technical details. Example of PIM: Formal definition of an operation that transfers funds from a checking account to a saving account. Specification that depends on interfaces and artifacts of DCOM for this operation would be DCOM specific. Specification that depends on interfaces and artifacts of CORBA for this operation would be CORBA specific. These are examples of PSMs.

MDA visions and standards

The vision of MDA is to create a development environment with efficient and nearly seamless interoperability between diverse applications, tools and databases. Components participating in the environment leverage standard services through the implementation of MDA standards. MDA standards enable such components to expose and interchange their metadata as instance

of well-defined models. The platform services have standard definitions that are expressed via standard programming models (APIs).

The key standards that make up the MDA include: UML, Meta Object Facility (MOF), Common Warehouse Metamodel (CWM), and XML Metadata Interchange (XMI) [11].

- UML: In 1997, the OMG has adopted the UML as a standard modeling language to be used as a modeling tool for object oriented software systems. Since then UML has gained popularity and support from the industry and tool vendors. UML is programming language independent. Its use cases and interaction diagrams have helped users to understand the analysis of models developed by system modelers. In this work, the UML is used as a graphical language for designing PIM, PM and PSM.
- MOF: It provides a framework for managing any type of metadata. The MOF has layered metadata architecture with a meta-modeling layer and an object modeling language closely related to UML that ties together the metamodels and models. The MOF also provides a repository to store metamodels.
- CWM: It provides standard interfaces that can manage many different databases and schemas throughout an enterprise. The CWM interfaces are designed to support management decision making and exchange of business metadata between diverse warehouse tools to help present a coherent picture of business conditions at a single point in time.
- XMI: It is used for representing and exchanging CWM metamodels using the Extended Markup Language (XML).

The MDA Process

According to [15, 12], the process of MDA has the following major steps for the enterprise system:

- specifying a system independently of the platform (PIM) (middleware, OS, programming language, communication protocol or hardware) that supports it,
- specifying the different target platforms (CORBA, DCOM….),
- choosing a particular platform for the PIM, and
- transforming the system specification (PIM) into PSM for the selected platform. PSM is a specification that captures the platform behavior, constraints and interactions with the target platform.

- transforming the PSM into the code according to the programming languages supported by the platform. The PSM carries all the necessary details for the coding.

Abstractions, Viewpoints and Refinements

A model from a given view point can be defined as a model that is created based on specific abstraction criteria. Abstraction is used to extract things to be included in the model at a higher level and to hide details from a model. However, refinement is the process of creating the lower level details from a higher level abstraction. Each element of a higher level model may be mapped to one or more elements of a lower level refinement. PIM and PSM can be considered as views of a system at different abstraction levels [22].

Models and Metamodels

A precise definition of a model, as used in the MDA [18], is: "A representation of a part or the complete functionality, structure and/or behavior of a system as seen from a particular viewpoint, described in a language which is well-formed in syntax and semantics". Examples of Models: Blueprints of a house, city road-maps.

In the MDA paradigm, application developers capture integrated, end-to-end views of entire applications in the form of models, including the interdependencies of components. This paradigm can then be used by domain experts to create the models and thus the applications. In addition to the concept of models, the MDA uses the concept of metamodels. Metamodels specify the syntax and semantics of models. Similarly, the metametamodels can be used to specify the syntax and semantics of metamodels.

A model can be seen as an instance of a metamodel while a metamodel can be seen as an instance of a metametamodel. This "instance of" relationship can be applied to create as many metalevels as needed. The MDA defines only four layers that are organized with meta relationship among them [17, 28] as shown in figure 2.3. The level M0 corresponds to a system which will be modeled using user model (i.e. M1).

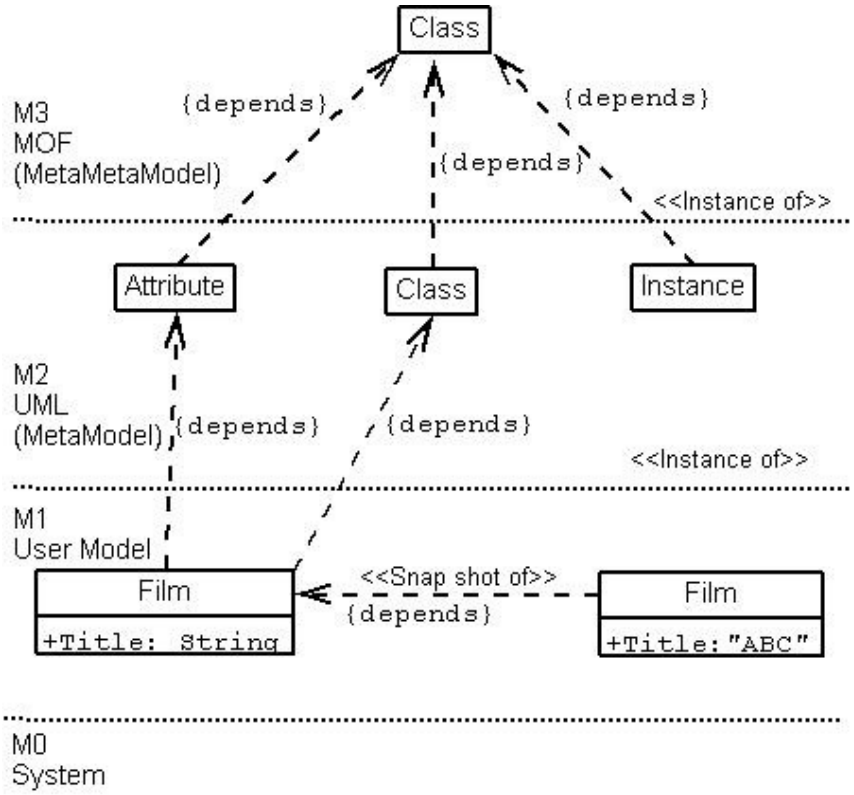


Figure 2.3 **An Example of Metamodel hierarchy in MDA**

In MDA metamodeling layered structure, the highest level (i.e. metametamodel) is described and defined using the MOF. The MOF is reflective in that it is defined using itself [17]. The models at metamodel (M2) level define an abstract language that is used to create models at (M1) level. The number of constructs decreases from level M1 to level M3. For defining models at metamodel level, there are two options. One choice is to define a completely new metamodel based on the MOF constructs as it is done for CWM in contrast to UML or extending an existing metamodel to provide the desired modeling capabilities as it is true for UML. The UML profiles are examples of the UML extensions defined for different application domains such as real-time, embedded systems and QoS [27].

Mapping and Transformation

One of the fundamental ideas of the MDA approach is to provide the ability to support software throughout its life time. For example, whenever new platform emerges, one has to decide whether or not to adopt the new platform and port his software to it. This decision can be very expensive, mostly because of the investment needed to port software to other platforms. In addition, not adopting this new platform may lead to considerable loss of income in case the old platform becomes obsolete.

The above scenario is just one example of the problems the MDA wants to address. In this particular example, the use of model transformation would reduce the investments necessary to port the software to new platforms. MDA addresses platform variability and evolution by the use of model transformations that would reduce the investments and time required to port the software to the new platform. Model transformation is the process of relating and mapping elements of one model into corresponding elements of another model. [15, 18] propose the usage of OMG core technologies to express model transformations. However, there is no standard mechanism for defining model transformations. The OMG has issued an RFP (Request for Proposal) for the design of the Query/View/Transformation specification [16]. This specification is expected to provide a standard mechanism for obtaining information models and defining transformations using the MOF [22].

Mapping: An MDA mapping provides specifications for transformation of a PIM into a PSM for a particular platform. The PM will determine the nature of the mapping. For instance, the QoS requirements on the implementation platform can be used as the mapping strategy to guide transformation. The mapping strategy will choose a target platform appropriate to those requirements [35].

Transformation: Model transformation is the process of converting one model to another model of the same system. This can be done manually or automatically.

- Manual Transformation: In order to make the transformation from PIM to PSM, design decisions can be made during the process of developing a design that conforms to engineering requirements on the implementation platform. For example, a whole range of QoS requirements can be used to guide transformation. Specific transformation choices will be made according to the particular qualities required at each conformance point in the mapping layer.
- Automatic Transformation: It is used for mature component-based development, where middleware provides a full set of services, and where the necessary architectural decisions are made once for a number of projects. These decisions are implemented in tools, development processes, templates, program libraries, and code generators. In such a context, it is possible for an application developer to build a PIM that is complete as to classification, structure, invariants, and pre- and post-conditions. The developer can then specify the required behavior directly in the model. This makes the PIM

computationally complete; i.e. the PIM contains all the information necessary to produce computer program code.

MDA Success and Benefits

OMG standardizes MDA modeling definitions and specifications. Many generic and domain based modeling specifications have been released. Some software development companies and business organizations are using the MDA technology as a new and successful approach in their system development process and these are confirming the MDA approach benefits [16]. When implemented properly with appropriate tools, MDA technologies help to:

- free application developers from dependencies on particular software APIs, which ensures that the models (PIMs) can be used for a long time, even as existing software APIs become obsolete and replaced by new ones.
- analyze the models automatically and offering refinements to satisfy various constraints.
- rapidly prototype new concepts and applications that can be modeled quickly using this paradigm, compared to the effort required to prototype them manually.
- save companies and projects significant amounts of time and effort in design and maintenance, thereby also reducing application time-to-market.

UML Extensions

The MDA key standards are extensible to fulfill the special nature of the embedded world and to be implemented in the real time and embedded applications. The advanced features in the modern embedded systems can also lead to an approach that makes the MDA an appropriate methodology for the development of embedded systems. From the MDA standards, most of the extensions are made to the UML. The UML is extensible through stereotypes, constraints and tagged values. The major extension mechanism is stereotype and it is a model element that extends another model element from its base. Stereotype makes the meaning and the usage of the extended element different from the base. Examples: business object and business process can be used by business modelers as different model elements with different attributes, operations and usage [1]. The extensions to the original UML specification are documented in UML profiles. The domain of real-time and embedded systems is one of the domains that is benefiting from the UML Extensions [24].

The UML contains some capabilities to support real-time aspects: either for qualitative aspects such as concurrency (Active objects, concurrent states, etc.) or for quantitative aspects such as time event. Nevertheless, these real time features of the UML are not enough. For that reason, OMG has initiated a work dedicated to define UML profile for Scheduling, Performance and Time Specification (SPT), a UML profile specific to real-time systems development [4]. SPT defines standard paradigms of use for modeling of time-, scheduling-, and performance-related aspects of real-time systems. The SPT specification defines (as a meta-model) a complete, but generic model of some of the key concepts association with scheduling, performance modeling and times events. Main concepts introduced in the SPT profile are QoS and Resource [19].

The QoS profile provides a mechanism to specify and quantify QoS characteristics (parameters). QoS characteristic is a quantifiable model property and it serves as the constructor of non-functional QoS parameters. QoS characteristic is quantified using QoS dimensions. Example: the rate dimension is used for the throughput. Minimum, maximum and average dimensions are used to quantify delay and priority. Direction is used as an attribute of QoS characteristic to indicate whether large values or small values are better. For example considering priority, decreasing its direction attribute value so that smaller priority values are better. In general, the direction increasing implies that large values are better while the direction decreasing implies that small values are better. Unit is used as attribute to express the values of QoS parameters. Example: bps is used for the dimension rate. The QoS constraint provides means to limit the possible values of QoS characteristic [19, 27].

The generic concept of QoS Constraint is refined into the following concepts:

- QoS required: this ensures a user to specify which QoS it requires from a provider when operating the required service.
- QoS offered: this ensures to define the QoS associated to services provided by model element.
- QoS contract: when linking a user and a provider, respective required and offered QoS has to match.

SPT profile introduces General Resource Modeling (GRM) framework. In this profile, a resource is defined as an entity on an execution platform (software or hardware) that offers one or more services with which a QoS is associated. The SPT profile classifies resources of

an execution platform as processor resource (CPU, memory), communication resources (protocols, mediums) and devices resources (Input/Output).

For this work, the integrated profile of QoS and Resource is used as a framework for modeling platform elements of PIM, PM and PSM that can describe the functional services and non-functional QoS characteristics. For this research, the following integrated profile of QoS and Resource (refer to figure 2.4) is used as the framework [27]. As indicated in the figure 2.4 below, there is only one instance of PIM where as there are as many PM instances as the number of mapping model instances.

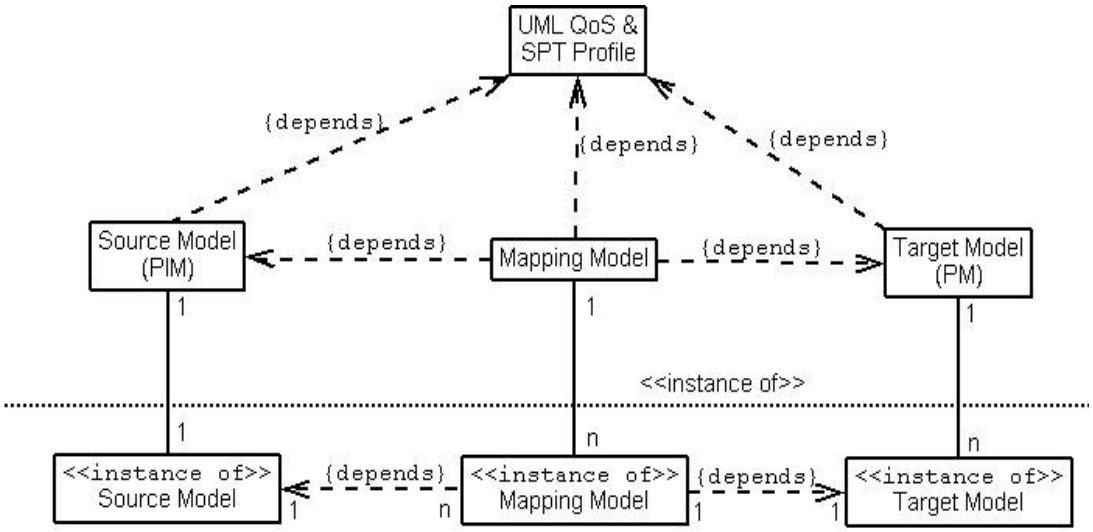


Figure 2.4 The Integrated QoS and Resource Profile

The summary of the UML extensions (stereotypes) introduced within these profiles [7] are listed in the table 2.1 below:

Stereotype	Base Class (UML core)	Comments
GRMResource	Class, classifier, Model Element	Represents a resource
QosCharacteristic	Class, classifier	Represents a quantifiable QoS characteristic associated with a resource
QoSDimension	Feature	Means of quantification
QoSConstraint	Model element	A general constraint expression involving one or more QoS characteristics
QoSOffered	Model element	Extends the QoS constraint
QoSRequired	Model element	Extends the QoS constraint
QoSValue	Instance Specification	A value associated with an instance of a QoS characteristic

Table 2.1 **The UML extensions (stereotypes)**

CHAPTER 3. I^2C and CAN Networks

In this section, the I^2C and CAN networks will be discussed. Both of them use serial buses. Although serial buses don't have the throughput capability of parallel buses, they do require less wiring and fewer IC connecting pins. However, these serial buses are not merely interconnecting wires; they embody all the formats and procedures for communication within the system. Devices communicating with each other on a serial bus must have some form of protocol that avoids all possibilities of confusion, data loss and blockage of information. Fast devices must be able to communicate with slow devices. The system must not be dependent on the devices connected to it; otherwise modifications or improvements would be impossible. A procedure also has to be devised to decide which device will be in control of the bus and when. And, if different devices with different clock speeds are connected to the bus, the bus clock source must be defined [23, 25].

3.1. I^2C

I^2C is an acronym for Inter-IC Communication. Its name literally explains its purpose: to provide a communication link between Integrated Circuits.

3.1.1. Introduction

Philips developed a simple bi-directional 2-wire bus for efficient inter-IC control. This bus is called the Inter IC or I^2C-bus. At present, Philips' IC range includes several CMOS and bipolar I^2C -bus compatible types. All I^2C -bus compatible devices incorporate an on-chip interface that allows them to communicate directly with each other via the I^2C -bus. This design concept solves many interfacing problems encountered when designing digital control circuits. I^2C is designed for low-cost, medium data rate applications. Devices on the bus can be considered as masters or slaves when performing data transfers. A master is the device which initiates a data transfer on the bus and generates the clock signals to permit that transfer. At that time, any device addressed by the master is considered as a slave. Masters are generally microcontrollers and slaves can be CPU, LCD, driver, memory and other chips [8, 25].

The I^2C bus is a half-duplex, multi-master bus requiring only two signal wires: a serial data line (SDA) and a serial clock line (SCL). These lines are pulled high via pull-up resistors and controlled by the hardware via open-drain drivers, giving a wired-AND interface. I^2C uses an addressable communication protocol that allows the master to communicate with individual slaves using a 7-bit address with the following format

7 Address bits	1 R/W bit

or using a 10-bit address with the following format

11110	2 most significant bits of Address	1 R/W bit	8 least significant bits of Address

A device that conforms to the 7 bit address, receives one address byte; the last bit of the address represents the R/W bit. A device that conforms to the 10 bit address receives two address bytes. The first byte consists of the extended addressing reserved address (11110) and the two most significant bits of the device address and the R/W bit. The second byte contains the eight least significant bits of the address.

During communication with slave devices, the master generates clock signals for both communications to and from the slave. Each communication begins with the master generating a start condition, an 8-bit data word, an acknowledge bit, followed by a stop condition or a repeated start. Each data bit transition takes place while SCL is low, except for the start and stop conditions. The start condition is a high-to-low transition of the SDA line while the SCL line is high. A stop condition is a low-to-high transition of the SDA line while the SCL line is high. The acknowledge bit is generated by the receiver of the message by pulling the SDA line low while the master releases the line and allows the line to float high. If the master reads the acknowledgement bit as high, it should consider the last communication word not received and take appropriate action, including possibly resending the data.

Some of the features of the I^2C -bus include:

- Maximum number of pins: 2. Only two bus lines are required; a serial data line (SDA) and a serial clock line (SCL)
- Maximum distance is 5m. At maximum distance, maximum speed is less than 10Kbps.

- Each device connected to the bus is software addressable by a unique address and simple master/slave relationships exist at all times; masters can operate as master-transmitters or as master-receivers
- It's a true multi-master bus including collision detection and arbitration to prevent data corruption if two or more masters simultaneously initiate data transfer
- Serial, 8-bit oriented, bi-directional data transfers can be up to 100 Kbps in the Standard-mode, up to 400 Kbps in the Fast-mode, or up to 3.4 Mbps in the High-speed mode
- Fixed-priority arbitration i.e. same order of resolution every time
- On-chip filtering rejects spikes on the bus data line to preserve data integrity

3.1.2. I^2C Characteristics

Both SDA and SCL are bi-directional lines, connected to a positive supply voltage via a current-source or pull-up resistor (refer to figure 3.1). When the bus is free, both lines are HIGH. The output stages of devices connected to the bus must have an open-drain or open-collector to perform the wired-AND function.

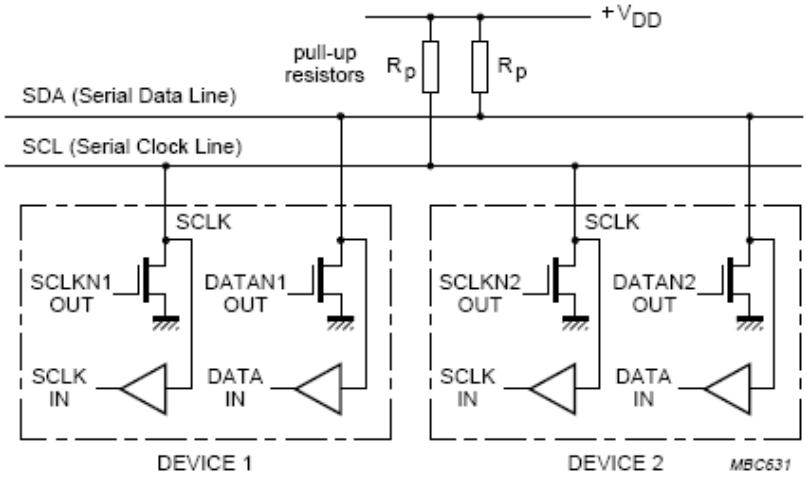


Figure 3.1 Connection of devices to I^2C-bus.

3.1.2.1. Bit transfer and data validity

I^2C bus master is responsible for generating SCL clock. Each master device must "listen" to the bus before transmitting to make sure that the bus is idle. Due to the variety of ICs (CMOS, TTL etc), which can be connected to the I^2C-bus, the levels of the logic '0' (LOW) and '1' (HIGH) that are transferred on the bus are not the same. CMOS input logic levels are 0v to 1.5v for logic 0 and 3.5v to 5v for logic 1 and the CMOS output logic levels are 0v to 0.1v for

logic 0 and 4.9v to 5v for logic 1. TTL input logic levels are 0v to 0.8v for logic 0 and 2v to 5v for logic 1 and the TTL output logic levels are 0v to 0.4v for logic 0 and 2.4v to 5v for logic 1.

The data on the SDA line must be stable during the HIGH period of the clock. The HIGH or LOW state of the data line can only change when the clock signal on the SCL line is LOW (refer to figure 3.2).

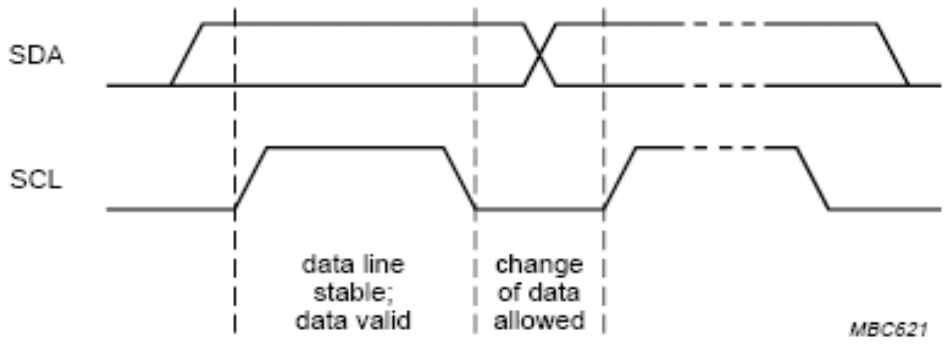


Figure 3.2 Bit transfer on the I^2C-bus.

3.1.2.2. Data Transmission

Figure 3.3 shows a simple communication exchange using a start bit to initiate communication, the 8 bit message, the acknowledge bit and then the stop bit. If additional data was to be sent, a restart bit could be sent instead of the stop. Bus transaction is a series of byte transmissions. It is initiated by a start signal and completed with a stop signal.

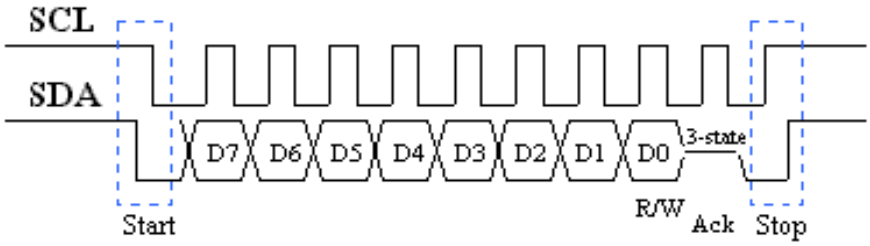


Figure 3.3 I^2C Data Transmission

The master devices generate the clock signal. The clock signal determines when data is transmitted, and how fast it is transmitted. The slave device must wait for the master device to initiate communication. Example: Assume PE1 is the master and PE2 is the slave. The PE1 wants to talk to one of its slaves - PE2.

- PE1 issues a 'START' condition: Start signal acts as an 'attention' signal to all of the connected devices. All devices on the bus will listen to the bus for incoming data.
- PE1 sends the address to the bus: At this moment all devices will compare this address with their own. If the address matches, the device will produce a response called the ACKNOWLEDGE signal. If it doesn't, they simply do nothing and wait until the bus is released by the 'STOP' condition.
- If PE1 gets this ACKNOWLEDGE then it can start transmitting or receiving data.
- When the transmission is completed, the master device will issue a STOP condition. This is a signal that indicates the bus has been released and the devices may expect another transmission to start at any moment.

3.1.2.3. I^2C Read and Write Operations

Devices in I^2C networks have addresses (7 bits in standard, 10bits in extension). Bit 8 of a 7 bit address signals read or write operation. An address can be followed by one or more data bytes.

(a) Write to a slave device: As indicated in the figure 3.4 below, the master is a master-transmitter and it transmits both clock and data during the entire data transfer.

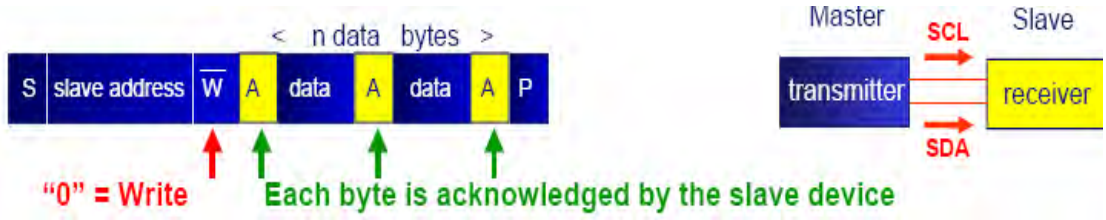


Figure 3.4 Write to a slave device

(b) Read from a slave device: As indicated in the figure 3.5 below, **t**he master is a master-transmitter initially then becomes master-receiver. It transmits clock all the time and it sends slave address and then becomes a receiver of the data bytes to be sent from the slave.

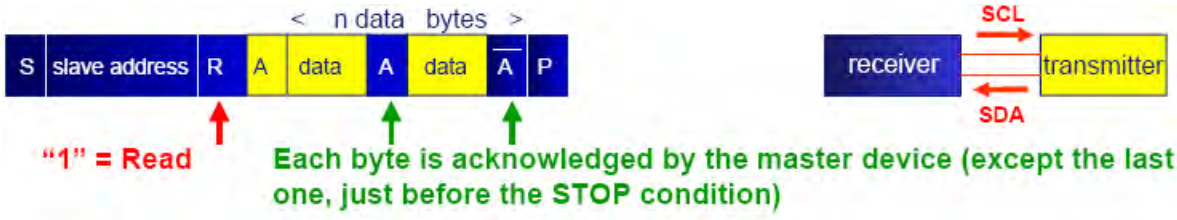


Figure 3.5 Read from a slave device

3.1.2.4. I²C Protocol Arbitration

Two or more masters may generate a START condition at the same time. Arbitration is done on SDA while SCL is HIGH; slaves are not involved in the arbitration. I²C Protocol uses fixed-priority arbitration; i.e. it always gives priority to competing devices in the same way. If a high-priority (lower address) and a low-priority (higher address) device both have long data transmissions ready at the same time - quite possible that the low-priority device will not be able to transmit until the high-priority device has sent all data packets. This is handled by the Bus Arbitration process to avoid the possibility of collisions on the I²C bus. If a device PE1 is a sender, it senses the bus to determine whether it is free or busy before sending the receiver's address. If its address is higher than the other sender's address, it stops signaling and gives up the control early enough in clock cycle. Receivers are always listening to the bus for their address. As indicated in the figure 3.6 below, master device PE2 (which sends DATA 2) wins the arbitration by pulling the SDA low first. Master device PE1 (which sends DATA 1) loses arbitration since the level of signal sent by PE1 is different from the level of signal on the SDA line.

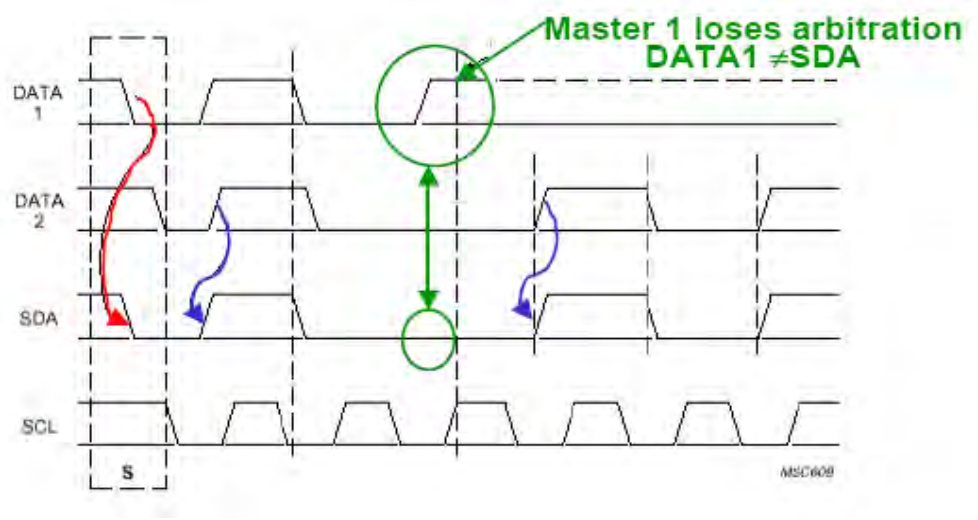


Figure 3.6 I²C Protocol Arbitration

3.2. CAN

CAN is designed to provide robust serial communications in automotive electronics. Bit-serial transmission is up to 1Mbps. CAN network uses a node-based approach: Information to or from devices (sensors, actuators, motors, host CPU, etc) is transmitted or received via a dedicated CAN node.

3.2.1. Introduction

CAN is a serial communication protocol. It is a broadcast bus with a multi-master architecture capable of providing real-time and fault tolerance features extremely relevant for control and automation. Because of its real-time and fault-tolerance capabilities, CAN has gained a wide acceptance in a large number of application areas. Its domain of application ranges from high speed networks to low cost multiplex wiring. In automotive electronics, engine control units, sensors, etc. are connected using CAN with bit rates up to 1 Mbps. At the same time, it is cost effective to build into vehicle body electronics such as lamp clusters, electric windows etc. to replace the wiring harness [8, 9].

The acceptance and introduction of serial communication to more and more applications have led to requirements that the assignment of message identifiers to communication functions be standardized for certain applications. These applications can be realized with CAN more comfortably since the original address range defined by 11 bits identifier in the standard format has extended to 29 bits identifier to provide larger address range in the extended format. This will relieve the system designer from compromises with respect to defining well-structured naming schemes. The extended format has been defined so that messages in standard format and extended format can coexist within the same network.

The CAN protocol is responsible for controlling the framing, performing arbitration, error checking, error signaling and fault confinement. It also deals with timing, encoding and synchronization of the bit stream to be transferred. Most of existing CAN implementations uses a differential two-wire bus line (the differential signal on the bus is used for noise rejection) as indicated in figure 3.7 below. Twisted pair/ optical link can be used as transmission medium for CAN protocol. The maximum length of transmission media depends on the data rate. Typical values are: 40m @ 1 Mbps; 100m @ 500 Kbps [23].

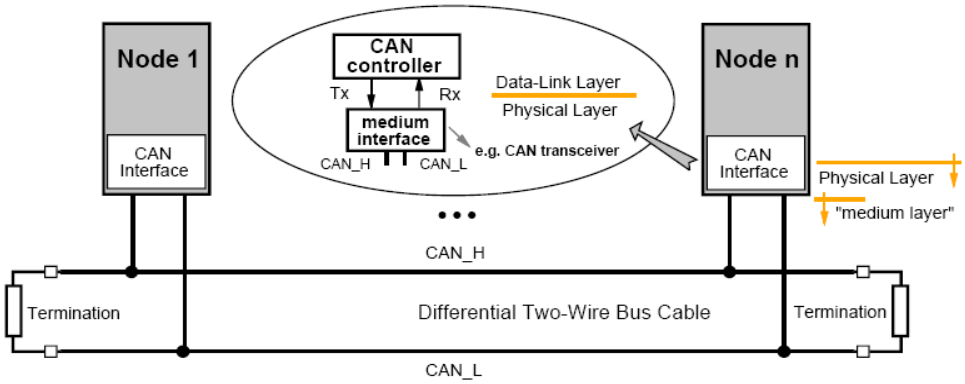


Figure 3.7 CAN node architecture

Bit signaling on the bus line can take two possible representations: recessive and dominant, In the wired-AND implementation of the CAN bus, the 'dominant' level would be represented by a logical '0' and the 'recessive' level by a logical '1'. This means that a dominant bit sent by one node can overwrite recessive bits sent by other nodes. This feature will be exploited for bus arbitration.

The data to be transferred is encapsulated within frames and a unique identifier is assigned to each one of these frames ("communication objects"). The uniqueness of communication object identifiers avoids the need of using addressing information. Distinct frames are used to provide (data frame) and to request (remote frame) the dissemination of communication objects. Two different lengths can be used for frame identifiers:

- ❖ Standard 11-bit identifiers (CAN 2.0A);
- ❖ Extended 29-bit identifiers (CAN 2.0B).

The uniqueness of communication object identifiers is also used to arbitrate bus access requests from competing nodes. CAN is a Carrier Sense Multi-Access with Deterministic Collision Resolution (CSMA/DCR) network: nodes delay transmissions if the serial bus-line is busy; when a bus idle condition is detected, any node may start transmitting; bus access conflicts are resolved through the bit wise comparison of communication object identifiers and it works as follows:

- ➢ While transmitting a communication object identifier, each node monitors the serial bus-line;

- If the transmitted bit is recessive and a dominant bit is monitored, the node gives up transmitting and starts to receive incoming data;
- The node transmitting the object with the lowest identifier gets access to the bus and starts its transmission.

That means: arbitration is non-destructive, since transmission of the object with the lowest identifier undergoes without delay; bus access is prioritized, allowing transmission of more urgent data to takeover less urgent one. Automatic retransmission of a communication object is provided after a loss in an arbitration process.

3.2.2. CAN Characteristics

3.2.2.1. Frame Formats

As mentioned above, two different lengths are allowed for data/remote frame identifiers. In order to maintain compatibility between these two definitions (standard and extended), distinct frame formats are used. The structure of a standard CAN frame format (version 2.0A) is shown in Figure 3.8 and the explanation of each field is given below [13, 26].

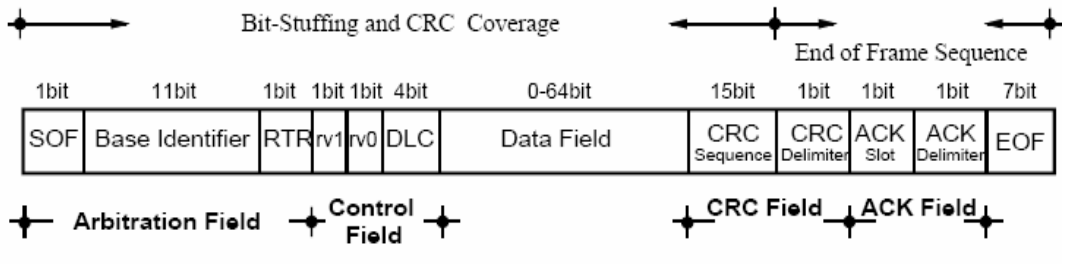


Figure 3.8 CAN 2.0A Data Frame Format

- **SOF** - Start Of Frame Delimiter - this field signals the beginning of the frame through the transmission of a single dominant bit (it means logical 0 in the wired-AND implementation)
- **Base ID -** the standard 11-bit sequence that uniquely identifies each communication object.
- **RTR-** Remote Transmission Request - this single bit field distinguishes data and remote frame types. RTR bit takes a dominant value for data frames; for remote frames this bit is set to recessive. Both the Base ID and the RTR field participate in the

arbitration process. Therefore for the same Base ID, data frames will take precedence over remote frames.

- **Control** - this field is six bits long and begins with two reserved bits that take always dominant values. It is followed by a four bit Data Length Code (DLC) that may assume any value in the interval [0, 8] and indicates the number of bytes in the Data Field. If the message is used as a remote frame, the DLC contains the number of requested data byte.
- **Data Field -** this field will hold the data to be transferred. Its size varies between zero and eight bytes.
- **CRC Field –** Cyclic Redundancy Code. This field consist of the following two elements:
  - **CRC Sequence -** a 15-bit, which is used by receivers to check the integrity of incoming frames.
  - **CRC Delimiter -** a single bit that is always set to a recessive value.
- **Acknowledge Field** – is used by receivers to acknowledge a correct data/remote frame transmission. It contains two distinct elements:
  - **ACK Slot -** a single bit element that is sent with a recessive value by the transmitter; its value is changed to dominant, by any receiver, upon reception of a frame without CRC errors.
  - **ACK Delimiter -** a single bit delimiter that takes always a recessive value.
- **EOF -** End Of Frame Delimiter **–** consists of seven recessive bits.

The size of CAN frame identifiers was extended in CAN Specification 2.0B to a length of 29 bits and its compatibility with the previous definition for CAN 2.0A was maintained. Let us now analyze the impact of such extension on data/remote frame structure (refer to figure 3.9). The two formats are distinguished through a different utilization of the two bits that immediately follow the standard 11-bit base identifier. In the standard 2.0A format definition, the first one of these two bits corresponds to the RTR field. In the extended 2.0B format it is replaced by:

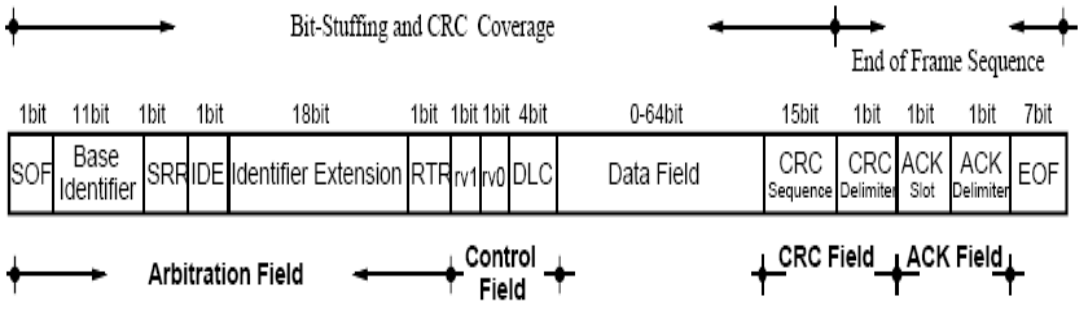


Figure 3.9 **CAN 2.0B Data Frame Format**

- **SRR** - Substitute Remote Request - a single bit that is always transmitted with a recessive value. Since the bit transmitted at this location in the standard CAN frame structure belongs to the arbitration field, which means CAN 2.0A traffic always prevails over any CAN 2.0B frame using the same 11-bit identifier.

The next bit to be transmitted belongs to the Control Field in the standard CAN frame structure; it is always sent with a dominant value. In the extended CAN frame structure the SRR bit is followed by:

- **IDE -** Identifier Extension - a single recessive bit is sent after the SRR bit in order to signal the presence of an identifier extension. It distinguishes standard frame from extended frame.
- **Extended ID -** An 18-bit sequence of additional identifier bits. It follows the IDE bit and precedes the RTR bit and the Control Field.

IFS -Intermission Frame Space- is the minimum number of bits separating consecutive messages. If there is no following bus access by any station the bus remains idle.

Arbitration on Message Priority: Ensuring that higher priority messages are transmitted ahead (bitwise arbitration). To provide deterministic prioritized medium access; bits on bus are either recessive (high) or dominant (low); the first node to transmit a dominant bit during arbitration gets the bus. If a node hears a 0 at the time it is transmitting a 1 then it stops transmitting. The node with the most leading 0's wins arbitration.

3.2.2.2. Frame Coding

In order to avoid the transmission of long sequences of bits with identical polarity, outgoing data/remote frames are subject to a bit-stuffing methodology that prevents more than five consecutive bits of identical polarity to be transmitted, through automatic insertion of a complementary bit. The bit-stuffing is performed from the start of frame delimiter until the end of the 15-bit CRC sequence. The CRC delimiter, the acknowledge field and the end of frame delimiter have a fixed form and they do not subject to bit-stuffing. The use of bit-stuffing makes frame transmission durations dependent not only on its size but also on its bit pattern.

3.2.2.3. Frame Timing

Several network parameters bound the time required to perform a given frame transmission, once the node gets access to the network after winning an arbitration process. A fundamental parameter that influences the duration of a given frame transmission is:

- **Data Rate** - The nominal rate of data signaling, on the bus. It gives a meaning to t_{bit}, the nominal duration of a single bit (it is 1 divided by the transfer rate in bps).
- Other parameters are the frame format specification (2.0A or 2.0B), the frame type (data or remote), the data field size (the data field size is always zero for a remote frame).

3.2.2.4. Error Handling

An extensive set of error detection methodologies are used in CAN in order to enforce the reliability of communication object transfer. Furthermore, as a general rule, errors are signaled as soon as they are detected. Error signaling is performed in CAN through the broadcast of special purpose data-link packets.

- **Error Signaling:** Most of the abnormal network operating conditions are signaled, by the node detecting such condition, through the issuing of an error frame. An error frame (refer to figure 3.10) begins with the transmission of an error flag and ends with an error delimiter. This last element always consists in a sequence of eight recessive bits. On the other hand, the error flag can either comprise a sequence of six consecutive dominant bits (active error flag) or a sequence of six consecutive recessive bits (passive error flag), depending on the node error status. Error signaling is performed as follows:

- ❖ The node detecting the error and initiating the error signaling starts it by transmitting one error flag;
- ❖ If there are nodes that have not yet detected the error, the transmission of that error flag will, in most cases (Exception made when a passive error flag is issued by a receiver), lead to an error condition (for instance when error flag pattern violates the bit-stuffing rule);
- ❖ All nodes respond to an error by starting their own transmissions of an error flag;
- ❖ At most, two error flags will flow in the network before the joint transmission, by all the nodes, of the corresponding ending delimiter (refer to figure 3.10).

Error frames are not subject either to CRC checking or to bit-stuffing.

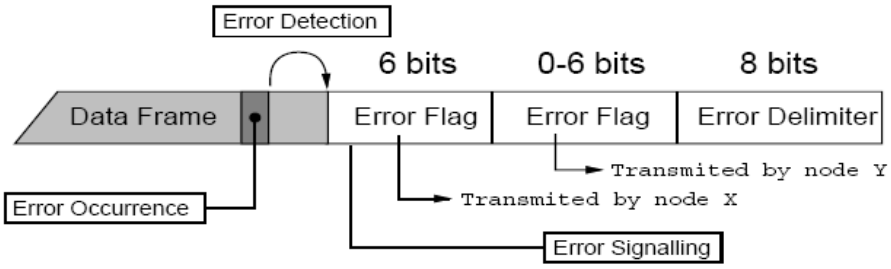


Figure 3.10 CAN Error Frame Format

- ➢ **Error Detection:** A set of different mechanisms are provided in CAN for error detection. These mechanisms include:
  - ❖ Bit Errors: each node listens to the bus while transmitting and compares both streams on a bit by bit basis. When the transmitted bit level does not match the monitored one, error signaling is started at the next transmitted bit.
  - ❖ Stuff Errors: a sequence of more than five consecutive bits with identical polarity is detected in a data or remote frame, within the corresponding bit-stuffing range. Transmission of an error flag is started immediately after detection of the bit-stuffing violation.
  - ❖ CRC Errors: when a CRC error is detected, the receiver takes no action at the acknowledge slot and starts error signaling only at the bit following the acknowledge delimiter, unless an error frame due to another error condition has already been started.

- ❖ Form Errors: a receiver detects a value violation in a fixed form frame field. This includes: the 1-bit CRC delimiter, the 1-bit acknowledge delimiter and the 7-bit end-of-frame delimiter, in data and remote frames; the full error and overload frames. The transmission of the corresponding error flag is started immediately.
- ❖ Acknowledge Errors: a transmitting node does not get any acknowledgment from the receivers. Recognition of the error is restricted to the transmitting node. Error signaling is started at the acknowledge delimiter.

Besides the aforementioned errors, there is another set of conditions that prevent normal network operation during the time required for its recovery. They concern violation of the three recessive bit period (intermission) that precedes any data/remote frame transmission and are known, in CAN terminology, as overload conditions. Like other errors, overload conditions are also signaled to other nodes through the broadcast of a special purpose data link frame, known in CAN terminology as overload frame. Overload frames have the same structure as error frames (i.e. an overload flag followed by an overload delimiter) and are broadcasted through a similar method. The only difference regards the overload flag: it always consists of six consecutive dominant bits.

The need for overload signaling can be due to external reasons: detection of a dominant bit during intermission; a receiver detects a dominant bit at the last bit of the end of frame delimiter. Overload signaling can also be performed under "request", due to the need of an extra delay by the internal receiver circuitry. A fundamental difference between errors and overload conditions is that the later ones do not influence the operation of error confinement mechanisms.

3.2.2.5. Error Confinement

The error confinement mechanisms provided by the CAN protocol aims to distinguish between temporary errors and permanent failures as well as the shutdown of defective nodes. These mechanisms are based on two different error counters recording, at each node, transmit and receive errors. These counters have a non-proportional update method, with each error causing an increment larger than the decrement resulting from a successful data or remote frame transfer. The rules used in error counting have been defined in order that nodes closer

to the error locus will experience, with a very high probability, the highest error count increase. This way, disturbances due to a faulty node can be localized and their influence restricted, accordingly with the following classification:

- Error-Active - the normal operating state. Such a node is able to transmit and receive frames and fully participates in error signaling.
- Error-Passive - the node is still able to transmit and receive frames, but: after transmitting a data or remote frame the node requires an extra eight bit bus idle period following the intermission, before it can start a new transmission; error signaling is performed by issuing a passive error flag, meaning it has only guarantees to succeed if the node was transmitting.
- Bus-Off - a node in this state does not participate in any bus activity, being unable to send or receive frames.

A node enters the "error-active" state after power-on initialization with both error counters set to 0. It will become "error-passive" if any error counter exceeds 127. An "error-passive" node goes back to "error-active" when both counters are equal to or lower than 127. Conversely, an "error-passive" node goes "bus-off" after its transmitter error counter exceeds 255. A node in the "bus-off" state is allowed to become "error-active" after a reset (that sets to 0 both error counters) and after 128 occurrences of 11 consecutive recessive bits have been monitored on the bus line. Total residual error probability for undetected corrupted messages is less than message error rate * 4.7 * 10^{-11}.

3.2.2.6. Data Transfer Consistency

The set of error detection mechanisms aim to ensure consistency of data transfers among "error-active" nodes. Even under the assumption that no transmitter failure occurs, it is required an extra condition to ensure that behavior. Frame Validity Rule - for a data/remote frame transfer be considered valid by a transmitter; it is required that no error occurs until the end of the corresponding end of frame delimiter. On the other hand, a receiver considers that transfer is valid if there is no error until the last but one bit of the end of frame delimiter.

An error in the last bit of a frame seen only by a set of receivers but not by the transmitter, will lead to an inconsistent transfer. The above rule allows receivers to accept frames whose transfer is disturbed by such errors, thus ensuring consistency in the absence of transmitter

failure. However, because of the aforementioned rule, there is the possibility that duplicates of the same frame could be received, by the set of receivers detecting no error until the last but one bit of a given data/remote frame transfer. Furthermore, if after such an error, the transmitter fails before being able to successfully complete the frame transfer, that transfer will stay inconsistent. [23, 9].

CHAPTER 4. Conceptual Model Development

The MDA process in the enterprise systems development uses middleware and gives least consideration to QoS. However, for the domain of embedded systems, the middleware cannot be supported due to limited resources and does not generally fit this domain. QoS is a major issue to be dealt with so that the limited resources in this domain can be utilized efficiently. Thus, this thesis work proposes a conceptual model that employs QoS based MDA approach using the integrated QoS and Resource UML profile framework (refer to figure 2.4) for applications that use priority based embedded networks such as CAN and I^2C as target platforms [4].

The UML profile for QoS and resource introduced in chapter 2 is used to model the PIM, PM and Mapping layer and PSM of the conceptual model. The analysis of the priority based networks (I^2C and CAN) QoS parameters and model development are explained in this chapter. Further, in order to realize some of the supported QoS parameters for I^2C network that are used in the PM of the conceptual model, a specific application is developed using I^2C as the target platform and the details of the implementation will be presented in chapter 5.

In embedded application development, if target platform can be represented abstractly and standard mappings and transformations are defined, the implementation and reusability of the embedded applications on different target platforms will be simplified and enhanced. Moreover, the time-to-market for the embedded products will be improved. The PIM will be an abstract model that can be used to represent the applications independent of the details of the implementation platforms. Upon transformation, the abstract platform (PIM) and the characteristics of the target platform (PM) are mapped into a specific platform model (PSM) through a mapping layer [10]. As a result, whenever a new platform is targeted; a new specific platform model is created with a modified mapping layer that reflects the characteristics of the new platform (PM) and the functional and structural elements of the existing application (PIM). This shows that an application can target a new platform without being modified [3].

As indicated in figure 4.1 below, the following steps are followed to develop a conceptual model using a QoS based MDA approach for the priority based embedded network applications [2].

- Identifying and analyzing the QoS parameters for I^2C and CAN networks.
- Defining the functional and QoS elements of the target PM using UML concepts. The functional elements include the services provided by the network such as send, receive, error signaling and other control functions. The non-functional elements are QoS characteristics that are used to measure the performance of the functional elements.
- Defining the PIM that can appropriately abstract the services and QoS parameters of the target networks.
- Transforming the services and QoS requirements of the PIM according to the capacity of the target network to PSM based on the mapping procedure.

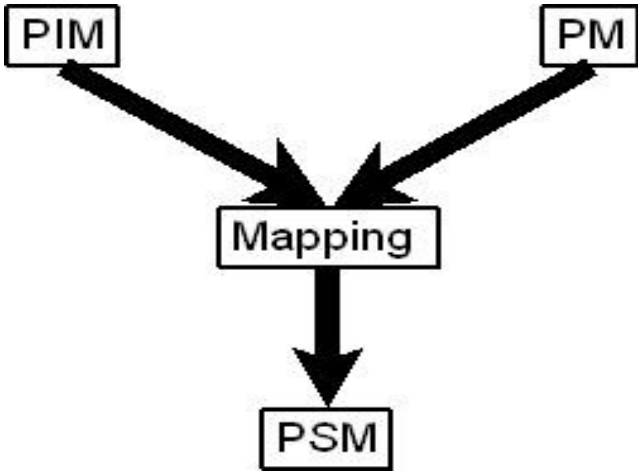
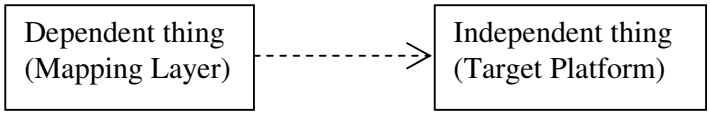


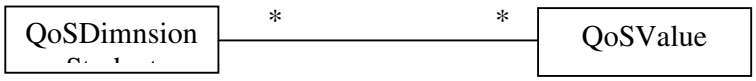
Figure 4.1 **General MDA process**

In the MDA approach, the collaborations between PIM, PM, Mapping and PSM can be represented using UML relationships such as Dependency, Association, Aggregation and Generalization [28].

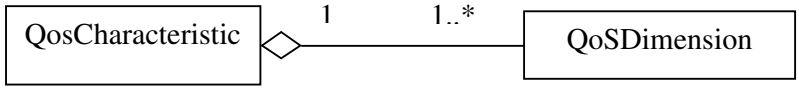
- **Dependency:** Given a dependency between two classes, one class depends on another but the other class has no knowledge of it. In dependency relationship, the dependent thing uses or depends on the independent thing. i.e. Change in independent thing causes change in dependent thing. Example: Change in a target platform causes change in the mapping layer and this can be indicated by using dependency relationship between them.



- **Association:** Given an association between two classes, both can rely on each other in some way. Association relationship indicates set of links (structural relationships) among objects (instances). Example: every QoS dimension (rate, minimum, maximum and average) may have different values. This can be represented by the following association relationship.



- **Aggregation** is a special kind of association (it has the implication of "has a") indicates aggregation relationship between things. Example: A QoS characteristics have one or more QoS dimensions and this can be represented using aggregation relationship as follows:



- **Generalization:** In the generalization relationship, a child inherits from its parents but the parent has no knowledge of its child. Generalization links general class (parent) to its specialization (child). Example: Request is a kind of Event can be represented by the generalization relationship as follows:

4.1. Identification and Analysis of network QoS parameters

4.1.1. Identification and Analysis of I^2C QoS parameters

In this section the functional and non functional QoS elements of I^2C will be identified and analyzed. Functional services include stream initialization (a phase that enables a user to initialize a communication endpoint for use in communication and establish an identity with the provider, this refers to the addressing process in I^2C network), data transfer (send, receive) and event signaling (start, stop and acknowledge signals). The major QoS criteria for I^2C based systems are: Delay, Priority, and Throughput [8, 26].

4.1.1.1. Delay

Delay on I^2C bus is given by the formula in Equation 4.1:

$$Delay = \frac{Frame\ \ Length}{BW} + Overhead\ \ \ Time \qquad (4.1)$$

Where:

- BW is the bandwidth of the I^2C bus.
- Frame Length represents sum of start bit, address bits, R/W bit, $8.L_D$ data bits, acknowledge bits, stop bit as indicated in figure 3.4.
  - L_D represents the number of data bytes to be transferred on the I^2C bus.
- Overhead Time = $t_x + t_r$
  - **t_x:** represents transmitter side overhead (instructions execution time to transmit a frame) and
  - **t_r:** represents receiver side overhead (instructions execution time to receive a frame).

Consider a case when one byte data transits (one data byte transfer) on 400 Kbps I^2C bus. To calculate the frame transit delay ($t_D = \frac{Frame\ \ Length}{BW}$), bit time and total length of a frame (the number of control bits and data bits) are required.

- The bit time ($t_{bit} = 1/BW$) is defined as t_{bit} =1/400kbps =2.5microseconds (µs).
- The frame length (l_{frame}) is the sum of 1 start bit, 7 address bits, 1 R/W bit, 1 acknowledgement bit, 8 data bits, 1 acknowledgement bit and 1 stop bit. I.e. l_{frame} = 20bits.

Therefore

$$t_D = l_{frame} \times t_{bit} = 20 \times 2.5\ \mu s = 50\ \mu s$$

The Overhead Time is affected by the number of instructions to be executed by the transmitter and receiver to send and receive byte(s). The frequency of the microcontroller to which I^2C will be interfaced also affects the Overhead Time. In the implementation to be presented in chapter 5, 225 instructions (each with two clock cycles) are executed by 7.373MHZ master-transmitter and slave-receiver microcontrollers for one byte data transfer. Overhead Time for different data bytes transfer includes the number of instructions for first byte transfer including addressing and control information (225 instructions) + 132 instructions for every

additional data byte transferred as indicated in Annex D. And also an assembly code, which is used to count the instructions, is included in the same Annex D. The 80C51 family instruction sets are indicated in Annex C.

Thus, the execution time for one instruction is equivalent to 2 cycles divided by 7.373 x 10^6cycles/s = 0.27126 µs. To execute 225 instructions, 225 x 0.27126 µs = 61.0335 µs is required. In this case, the overhead time: tx + t_r = 61.0335 µs for one byte data transfer. For every additional data byte transfer another 132 x 0.27126µs = 35.80632 µs is required.

For different number of data bytes transfer the corresponding total delay is given in the table 4.1 below.

Number of Data Bytes	$\frac{Frame\ \ Length}{BW}$ in µs	Overhead Time in µs
1	20 x 2.5 = 50	61.0335
2	29 x 2.5 = 72.5	61.0335 +35.8063
3	38 x 2.5 = 95	61.0335 +2 x 35.8063
4	47 x 2.5 = 117.5	61.0335 +3 x 35.8063
5	56 x 2.5 = 140	61.0335 +4 x 35.8063
6	65 x 2.5 = 162.5	61.0335 +5 x 35.8063
7	74 x 2.5 = 185	61.0335 +6 x 35.8063
8	83 x 2.5 = 207.5	61.0335 +7 x 35.8063

$Delay = \frac{F}{}$

Table 4.1 **Theoretical Delay Time on 400Kbps I^2C bus**

In the above table, the frame length for different data byte transfer includes the bits present in one byte transfer (20 bits) + number of extra data bytes to be transferred + 1 acknowledgment bit for every additional data byte transferred.

4.1.1.2. Priority

In multi-master I^2C communication, two or more masters may generate a START condition at the same time in order to access the I^2C bus. For a master with highest criticality in using the services and resources on I^2C network, it is possible to assign the highest priority using the lowest address for the slave with which the master wants to communicate. Thus, the utilization (QoS) of the required services and resources is enhanced for the highest priority master. The I^2C based experimental setup (refer to section 5.1) uses single master I^2C communication and hence the priority is not considered.

4.1.1.3. Throughput

Throughput is defined as the number of bits transiting on the I^2C bus during one second (bits/sec). Throughput can be obtained using equation 4.2 shown below.

$$Throughput = \frac{frame\ \ length}{Delay\ \ Time} \qquad (4.2)$$

For different number of data bytes transfer the corresponding throughput is given in the table 4.2 below.

Number of Data Bytes	Frame Length(in bits)	Delay in µs	$Throughput = \frac{frame\ \ length}{Delay\ \ Time}$ Kbps
1	20	111.0335	180.1258
2	29	169.3398	171.2533
3	38	227.6461	166.9258
4	47	285.9525	164.3630
5	56	344.2588	162.6683
6	65	402.5651	161.4646
7	74	460.8714	160.5654
8	83	519.1777	159.8682

Table 4.2 **Theoretical Throughput on 400Kbps I^2C bus**

4.1.2. Identification and Analysis of CAN QoS parameters

In this section, the functional and non functional QoS elements of CAN will be identified and analyzed. Functional services include stream initialization (a phase that enables a user to initialize a communication endpoint for use in communication and establish an identity with the provider via identifiers), data transfer (send, receive) and event signaling (start, acknowledge, stop, error signaling, error detection signals). The major QoS criteria for I^2C based systems are: Delay, Error Detection Delay, Priority, and Throughput [8, 13, 26].

4.1.2.1. Delay Time

The advantages of serial bus systems in industrial automation systems are well known. On the other hand, the serialization of the information causes a notable delay time in comparison to parallel wiring. This delay time depends on the length of the data protocol (frame length), the baudrate and on overhead time and it is given by equation 4.3 as shown below.

$$Delay\ Time = \frac{Frame\ Length}{Baudrate} + Overhead\ Time \tag{4.3}$$

The frame contains information to synchronize, to identify, to control and to save the data flow in addition to the user data. Therefore, not only the frame format (standard/extended) but also the length of the user data determines the frame length (refer to figures 3.8 and 3.9).
The frame length is the most important factor of influence on the delay time. The baud rate is another important factor that influences the delay time. It depends on the transmission distance [9].

The overhead time contains the software delay time, the controller delay time and the bus access time. Delay time is called the time from a data change in one bus node up to its registration in another bus node.

One part of the delay time is the frame delay time t_T. The frame delay time is determined by the frame length L_T and the baud rate d_U. Table 4.3 shows parts of CAN 2.0A Data Frame (refer to figure 3.8).

Meaning	No. of Bits	Comments
Start of Frame (SOF)	1	
Arbitration field	11+1	Arbitration field Identifier + RTR For Standard CAN
	29+1	Arbitration field Identifier + RTR For Extended CAN
Control field	6	two reserve bits + Data Length Code
Data field	0.. 64	0.. 8 Byte
CRC field	15 + 1	CRC sequence + CRC delimiter
Acknowledge field	2	ACK slot + ACK delimiter
End of Frame (EOF)	7	

Table 4.3 **CAN Data Frame**

According to the CAN specification, the identifier is 11 or 29 bits long. That means, up to 2,048 data objects for the Standard CAN and 536,870,912 data objects for the Extended CAN can be addressed. In addition to the bits in table 4.3, one has to consider the stuff bits. A stuff bit is added by the transmitter after 5 consecutive equal bit levels and is rejected by the receiver. This bit has the inverted bit level. It is used to identify a bus idle, to synchronize the CAN controller and to detect transmission errors. The bit stuffing method begins with the Start of Frame and ends with the CRC sequence. Because of the stuff bit method, the delay time of a CAN data frame t_T has a minimal and a maximal value. equation 4.4 applies to event driven bus access (CSMA/CA) for the Standard CAN specification[26].

$$\frac{19+8.L_D+25}{d_U} \le t_T \le \frac{19+8.L_D+25+0.2(19+8.L_D+15)}{d_U} \qquad (4.4)$$

Where:

- 19 represents the sum of: SOF 1-bit, (11+1) bits of Arbitration field for the Standard CAN specification and the 6 bits Control field.
- $8.L_D$ represents the length of data frame in bits.
- 25 represents the sum of: CRC field (15+1) bits, Acknowledge field 2 bits and EOF 7 bits.
- 15 reprsents the number of CRC field 15 bits(CRC sequence) that are subjected to bit-stuffing.

The maximal delay time occurs, if after each 5th bit in the bit stream, a stuff bit (refer to section 3.2.2.2) is added. That means, a 20% longer delay time will be experienced in comparison to the minimal value.

Similarly, equation 4.5 applies to event driven bus access (CSMA/CA) for the extended CAN specification to determine the delay time of a CAN data frame t_T, which has a minimal and a maximal value.

$$\frac{37+8.L_D+25}{d_U} \le t_T \le \frac{37+8.L_D+25+0.2(37+8.L_D+15)}{d_U} \qquad (4.5)$$

Where:

- **37** represensts the sum of: SOF **1-bit**, **(29+1) bits** of Arbitration field for the Extended CAN specification and the **6 bits** Control field.
- $8.L_D$ represents the length of data frame in bits.
- 25 represents the sum of: CRC field (15+1) bits, Acknowledge field 2 bits and EOF 7 bits.
- 15 reprsents the number of CRC field 15 bits(CRC sequence) that are subjected to bit-stuffing.

The maximal delay time occurs, if after each 5th bit in the bit stream, a stuff bit is added. That means a 20% longer delay time in comparison to the minimal value.

Besides the CAN data frames there are also CAN remote frames. In that case, a master in a CAN system must send a remote frame to another bus node to get the data.

Parts of a remote frame (refer to figure 3.9) are shown in table 4.4.

Meaning	No. of Bits	Comments
Start of Frame (SOF)	1	
Arbitration field	11+1	Arbitration field Identifier + RTR For Standard CAN
	29+1	Arbitration field Identifier + RTR For Extended CAN
Control field	6	two reserve bits + Data Length Code
CRC field	15 + 1	CRC sequence + CRC delimiter
Acknowledge field	2	ACK slot + ACK delimiter
End of Frame (EOF)	7	
Intermission	3	

Table 4.4 **CAN Remote Frame**

The remote frame contains no data. Thus, the delay time to get a data by sending remote frame to another bus node can be calculated with Equation 4.6 for the Standard CAN specification.

$$\frac{19+25+3+19+8.L_D+25}{d_U} \le t_T \le \frac{19+25+0.2(19+15)+3+19+8.L_D+25+0.2(19+8.L_D+15)}{d_U} \qquad (4.6)$$

Where:

- 19 represents the sum of: SOF 1-bit, (11+1) bits of Arbitration field for the Standard CAN specification and the 6 bits Control field (refer to table 4.4) for each frame (remote and data)
- $8.L_D$ represents the length of data frame in bits.

- 25 represents the sum of: CRC field (15+1) bits, Acknowledge field 2 bits and EOF 7 bits.
- 15 reprsents the number of CRC field 15 bits(CRC sequence) that are subjected to bit-stuffing.
- 3 represents the minimum number of bits separating consecutive messages.

Note: the (19+5) bits are for remote frame and the (19+8.L_D+25) bits are for data frame in the Standard CAN specification.

The maximal delay time occurs, if after each 5th bit in the bit stream a stuff bit (refer to section 3.2.2.2) is added. That means a 20% longer delay time in comparison to the minimal value.

Similarly, the delay time to get a data by sending remote frame to another bus node can be calculated with Equation 4.7 for the Extended Standard CAN specification.

$$\frac{37+25+3+37+8.L_D+25}{d_U} \leq t_T \leq \frac{37+25+0.2(37+15)+3+37+8.L_D+25+0.2(37+8.L_D+15)}{d_U} \qquad (4.7)$$

Where:

- **37** represensts the sum of: SOF **1-bit**, **(29+1) bits** of Arbitration field for the Extended CAN specification and the **6 bits** Control field (refer to table 4.4) for each frame (remote and data)
- 8.LD represents the length of data frame in bits.
- 25 represents the sum of: CRC field (15+1) bits, Acknowledge field 2 bits and EOF 7 bits.
- 15 reprsents the number of CRC field 15 bits(CRC sequence) that are subjected to bit-stuffing.
- 3 represents represents the minimum number of bits separating consecutive messages.

Note: the (37+5) bits are for remote frame and the (37+8.L_D+25) bits are for data frame in the Extended CAN specification.

The maximal delay time occurs, if after each 5th bit in the bit stream a stuff bit is added. That means a 20% longer delay time in comparison to the minimal value.

The following values in table 4.5 are obtained using equations from 4.4 and 4.5 with the baud rate of 500Kbit/s that is common for many industrial applications [9, 6].

	Identifier	Data Length [Byte] : L_D							
		1	2	3	4	5	6	7	8
Data frame min. t_T	11-bit-ID	104	120	136	152	168	184	200	216
	29-bit-ID	140	156	172	188	204	220	236	252
Data frame max. t_T	11-bit-ID	121	140	159	178	198	217	236	254.5
	29-bit-ID	164	183	204	222	241	260	279	298

Table 4.5 **CAN delay times in µs for 500Kbps**

The maximum frame delay time($\mathbf{t_T}$) shown in table 4.5 plus controller and software delay time can be used to determine total delay time of the object with the highest priority in a CAN system, for a given number of data byte transfer.

The **overhead** time due to controller and software delay time varies depending on the types of microcontroller, CAN controller, software, and other hardware that will be used to develop a CAN based embedded system.

4.1.2.2. Error Detection Delay

Error Detection Delay is equal to the maximum delay for the receiver of information (frame) to detect the off status of the information transmitter. It is composed of Transmitter Overhead Time, Receiver Overhead Time and delay on the CAN bus. This delay is currently known as timeout. Error frames (refer to figure 3.10) are not subject either to CRC checking or to bit-stuffing. The corresponding best and worst transmission times are given by the following equations 4.8 and 4.9:

$$t_{error}^{bc} = (l_{flag} + l_{del})t_{bit} \quad (4.8)$$

$$t_{error}^{wc} = (2 \times l_{flag} + l_{del})t_{bit} \quad (4.9)$$

Where:

l_{flag} and l_{del} are, respectively, the lengths in bits of the error flag and of the error delimiter.

t_{bit} is the nominal duration of a single bit.

Consider a 500Kbps CAN bus, the values of the worst case and best case Error Detection Delays are shown in the table 5.5 below. From the given baud rate (500Kbps) of the CAN bus the t_{bit} can be obtainded as: t_{bit} is given by 1/500Kbps = 2µs. Referring to figure 3.10, the l_{flag} = 6bits and the l_{del} = 8bits.

l_{flag}	l_{del}	t_{bit}	$t_{error}^{bc} = (l_{flag} + l_{del})t_{bit}$	t_{error}^{wc} =
6 bits	8 bits	2µs	(6+8) x 2µs = 28µs	(

Table 4.6 **Values of Error Detection Delay**

4.1.2.3. Throughput

Throughput is defined as the number of bits transiting on the bus during one second(bits/sec). The CAN communication protocol implies that the request data frame and the data frame are sent for each information every communication period p. Thus the throughput is calculated using equation 4.10:

$$Throughput = \sum_{i=1}^{N} \frac{size(req) + size(data)}{p} \quad (4.10)$$

Where:

size(req) = size of the request data frame (remote frame)

size(data) = size of the data frame

4.1.2.4. Priority

Arbitration on Message Priority: Ensuring that higher priority messages are transmitted ahead (bitwise arbitration). To provide deterministic prioritized medium access; bits on bus are either recessive (high) or dominant (low); the first node to transmit a dominant bit during arbitration gets the bus. If a node hears a 0 at the time it is transmitting a 1 then it stops transmitting. The node with the most leading 0's wins arbitration. According to the CAN specification, the identifier is 11 or 29 bits long. That means up to 2,048 data objects for the Standard CAN and 536,870,912 data objects for the Extended CAN can be uniquely identified.

4.2. PM for priority based embedded networks

The target platform model refers to a priority based embedded network model. In this model, the service interface, the functional elements and the model elements including QoS characteristics of a given priority based embedded network are defined using UML concepts.
Service Interface: represents an entity through which a user and a provider of the data link services can communicate. It is a collection of operations that specify a service of a given element.

The functional services are: Stream Initialization, Data Transfer (transmit, receive) and Event Signaling. The model elements are: Bus- the serial line through which data and control information can be transferred, Event- every message exchange is produced as an event to invoke corresponding operation and QoSSpec- represents QoS constraint for the priority based embedded networks. These model elements are represented by classes. The bus is considered as a resource and QoSSpec is taken as a QoSConstraint. The remaining elements are modeled using the core UML standards.

The PM is represented by the platform metamodel (UML) profile framework,as indicated in figure 4.2 below. This metamodel framework includes: Bus, Event, QoSSpec, QoSCharacteristics, QoSDimension, QoSValue and ServiceInterface. All priority based embedded networks' PMs will be specific instances of this metamodel framework.

As indicated in the figure 4.2 below, a service and event signaling(indication and request are kinds of event) on a given network (PM) uses the transmission bus as a resource and they are associated with the supported QoS specification of the network that has QoS parameters (with corresponding QoS dimensions) associated with QoS values.

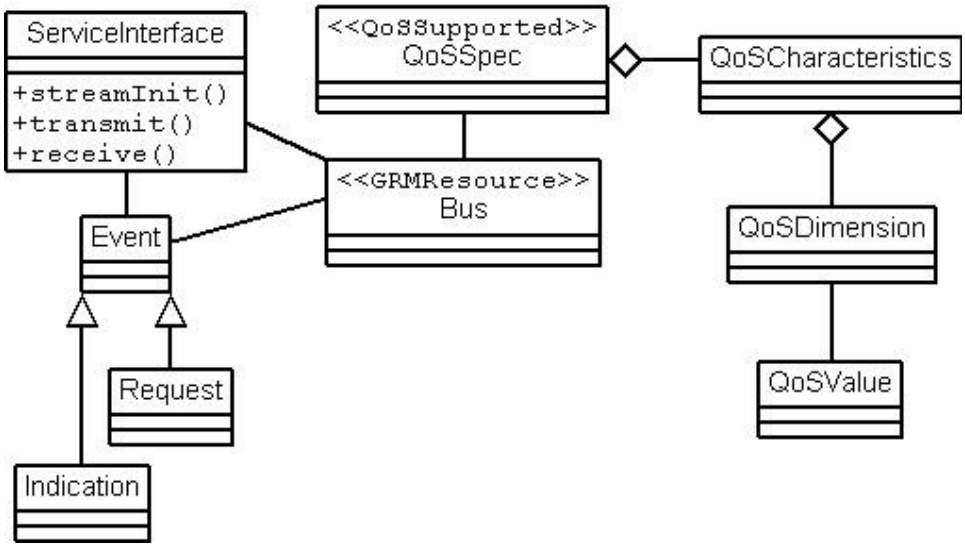


Figure 4.2 **The platform framework metamodel**

4.3. Platform Independent Model

The PIM acts as an abstract model that can be used within the model of the applications. Upon transformation, the PIM will be mapped to a PSM through a mapping layer. The applications will be implemented with the PIM specification on a specific target network. Whenever, a new network is targeted, only the mapping layer is modified so that the application still interacts with the mapping layer to be mapped to the new target network.

The main objective of the PIM is to represent the services of the priority based embedded networks in a generic manner so that applications deign and specification can be independent of the underlying network. With the MDA standard, the PIM should reflect characteristics of the PMs. This implies for priority based embedded networks, PIM should reflect their services and QoS characteristics. The applications implemented on these networks are aware of their services and QoS parameters. However, the applications design and specification will be independent of a specific network. It will be the responsibility of the mapping layer to transform this specification to the underlying network. Hence, producing a separation of concerns between the applications and the networks.

The PIM concepts are based on the following characteristics and they are represented using UML concepts as indicated in the figure 4.3.

- ServiceInterface is an abstract entity that represents the service interface where users interact with the network. This specifically abstracts the interface of the networks. It serves as a generic, abstract and high-level service interface for application modelers.
- Bus is a serial line through which communication occurs between a transmitter and a receiver.
- UDQoSSpec is the unit data QoS specification and it is an abstract representation of the QoS specification model element in the PMs. It is the specification of the QoS for each unit data transmission.
- Event represents all messages used as notifications for QoS parameter values selection, message sending and receiving, error detection and signaling, communication starting and stopping. It is an abstract representation of the event model element in the PMs.

UDQoSSpec has the following parameters: Delay, Priority, Throughput and Error Detection Time. It represents the QoS characteristics with its own quantifying dimensions and associated values. Since there is only one PIM, no metamodel representation is needed for the PIM.

As indicated in figure 4.3 below, the abstract service and event signaling that are associated with message exchanges at the application level interface (PIM) requires some QoS specification (Delay, Throughput, Priority, Error Detection Time) to be associated with them when they are using the transmission bus for data and control transfer.

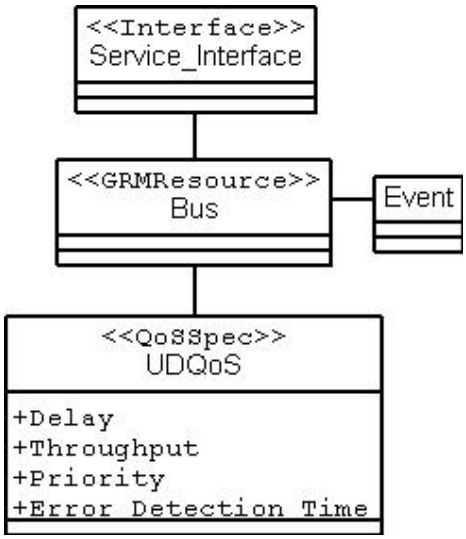


Figure 4.3 The Proposed PIM for priority based embedded networks

4.4. Conceptual Model

The conceptual model that includes the PIM, PM, Mapping layer and the PSM is given in the figure 4.4 as shown below. The PIM and PM are described in the above sections, and the mapping layer and the PSM will be presented below.

The mapping layer exits between the PIM generic layer and the PM specific layer. It is modified for each specific priority based embedded network it is targeting. The associations of service interfaces with the message exchanges are handled by the protocols in these networks and hence the service mapping aspect is not considered in this thesis work. The mapping strategy for QoS is the major mapping required to associate the QoS parameter values supported by a specific network to the QoS selected by the service user (PIM) in order to select the appropriate network that can be used for a final implementation.

The QoS parameter values supported by a specific network (PM) and the QoS parameter values selected by the service user (PIM) can be compared by the intermediate mapping layer to define a PSM. As long as an appropriate mapping layer has been defined from generic PIM specification to a specific network (PM). The objective of the MDA, which states that an application specification and its implementation can be separated as different concerns of development, is achieved using the mapping layer. The mapping layer takes the functional

services and non-functional QoS characteristics from a PIM and based on the capability of the target network (PM) it creates the PSM.

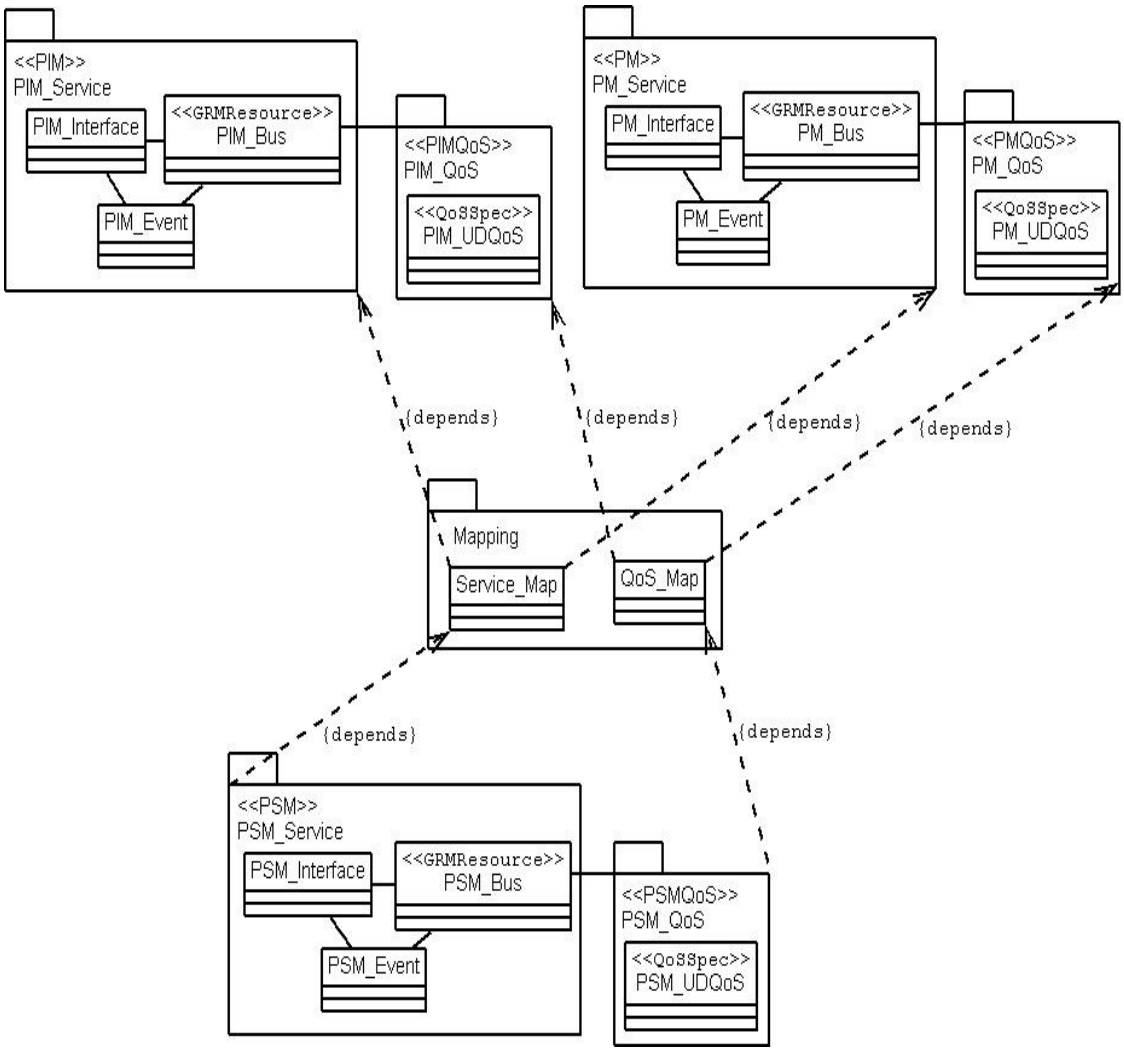


Figure 4.4 **The Conceptual Model**

4.4.1. The procedure for QoS Mapping.

In priority based embedded networks, the QoS applied to each data unit request may be independent of the QoS values to be associated with each transmission. The data unit QoS request may be issued to alter those values before each data unit request is issued. Using the information about the supported values of QoS parameters for given networks and the application QoS requirements, the mapping layer can determine the corresponding PSM. The PSM produced by this layer contains information about the appropriate network that can be used for the final implementation. And also the mapping layer determines the QoS parameters

(these QoS parameters information is contained in the PSM) that can be associated with each unit data transfer when the application gets implemented on the selected network .

Two special QoS values are defined for all the parameters used in QoS selection. These values are: **unkown** and **Don't_care** [5]. **Unkown** value indicates that the service provider (for a given network) does not know the value for the QoS field or does not support the specified QoS parameter. **Don't_care** value indicates that the service user (application specification in PIM) does not care to what value the QoS parameter is set. These two values are used in the proposed mapping procedure presented below.

In the mapping procedures for selecting QoS parameter values that will be associated with the transmission of unit data, the service user sends the selected QoS parameters and their values to the mapping layer:

- If the networks support no QoS parameters, the current QoS fields are set to **unknown** by the mapping layer. Then mapping layer notifies the service user about this situation and no PSM will be produced.
- If a network supports QoS parameters, the mapping layer notifies the service user the supported range of parameter values for delay, throughput, priority and error detection delay and produces the corresponding PSM that will be used for the final implementation.
- If the service user does not care what QoS is provided for a particular parameter, the corresponding QoS fields are set to don't_care by the mapping layer. The mapping layer retains the current value for that parameter and notifies the service user about this situation and PSM will be produced.

4.5. Conclusion

This chapter presented the main QoS based MDA approach for the priority based embedded network services, which includes both the functional and QoS characteristics. The QoS characteristics are based on the frame transmission delay, throughput, error detection delay and priority. PIM and PM modeling and a possible mapping procedure have been proposed. The major focus of the transformation is the QoS aspect. In a transformation to a PSM, specific network choices will be made according to the particular qualities required at each conformance point in the mapping layer.

A mapping layer compares PIM QoS requirements with the supported values of QoS parameters of a given network in order to determine whether a corresponding PSM will be produced or not. The mapping layer hides the details of a specific network from the application design and specification and hence it separates the two concerns of development in the domain of embedded system as application and network.

CHAPTER 5. Implementation

5.1. System Overview

As indicated in figure 4.4, the conceptual model consists of the PIM, PM, mapping layer and PSM. The application QoS requirements are specified using the PIM. The supported values of the QoS parameters by a given target network are specified using the PM. The applications QoS requirements are compared with the supported values of the QoS parameters by the mapping layer in order to produce the corresponding PSM. The PSM specifies the appropriate target network that will be selected as the target platform on which the application can be implemented.

The supported values of the QoS parameters for a given network are required in order to compare them with the application QoS requirements to determine whether the network can support these requirements or not. The following I^2C based application is used to measure the supported values of the QoS parameters such as delay and throughput for a 400kbps I^2C network at different I^2C bus lengths. The results from these applications can help the embedded application developers to choose this network based on their applications requirements for the QoS parameters and bus lengths.

If the mapping layer includes the supported values of the QoS parameters for different networks such I^2C, CAN and others in the domain of embedded systems, the embedded application developers can select the appropriate target network that can best fit their QoS requirements through the mapping layer.

And also the experiment is used to demonstrate a means of measuring and providing delay and throughput related information of this specific network at 50cm and 90.5cm bus lengths to the mapping layer. Then the mapping layer compares an application QoS requirements with supported values of delay and throughput of this network in order to determine whether the corresponding PSM will be produced or not. The PSM to be produced contains information about the application QoS requirements and the specific target network that fulfills these requirements. Then this network will be used for the final implementation.

The following hardware and software are used to implement the experimental setup.
The experiment is conducted using an I^2C bus and 89LPC932A1 microcontrollers (as master-transmitter and slave-receiver). And also, program loader, keil software and flash magic are used for this implementation.

For this implementation, a 400 Kbps I^2C network is used as the target platform. The microcontrollers used for this implementation have built in I^2C interface. The microcontrollers operate with internal oscillator circuits at 7.373 MHz. One machine cycle for these microcontrollers is equivalent to 2 clock pulses. The timer value in such microcontroller increments every machine cycle. Thus, **one increment in timer value** is equal to 2 divided by 7.373MHz. This value is equivalent to **0.27126 µs**. To obtain the measured delay time for a given number of data bytes transfer, the number of increments from the timer is multiplied by 0.27126 µs. The corresponding measured throughput is calculated by dividing the total number of bits (control bits: start bit, acknowledge bit(s) and stop bit, address bits, R/W bit and data bits) with the measured delay time for a given transfer.

Using the developed I^2C based Graphical User Interface (GUI) indicated in Annex A, the application user/programmer can choose the number of data bytes in order to determine the delay and throughput values that appropriately fits his/her application requirements for the given I^2C network.

The Experimental Setup for the I^2C based implementation consists of the following three modules as indicated in figure 5.1 below. The circuit diagrams for this implementation are indicated in the Annex B.

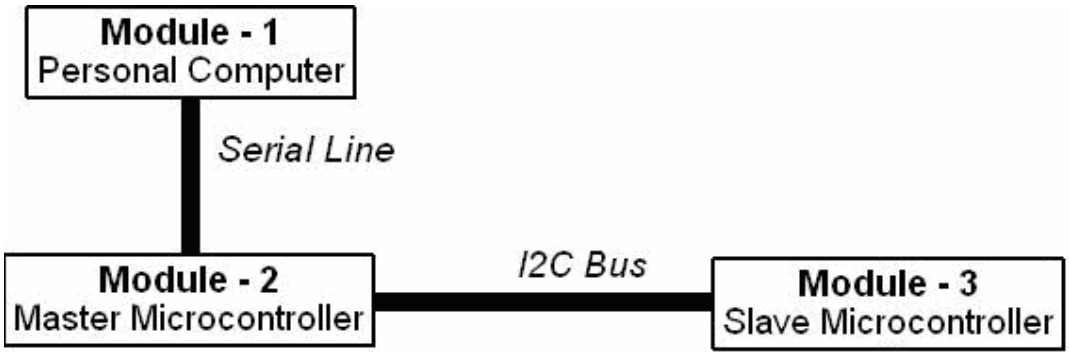


Figure 5.1 **The Block Diagram for Experimental Setup**

- Module1- personal computer helps to:
  - select the number of data bytes to be transmitted from the master microcontroller to the slave microcontroller through the I^2C bus.
  - display the timer value that helps to determine delay and throughput for the selected number of data bytes to be transferred between the master microcontroller and the slave microcontroller.
  - display the theoretical and measured values for the delay and throughput for the selected number of data bytes to be transferred.

- Module2- Master- microcontroller helps to:
  - receive the selected number of data bytes from the personal computer.
  - initiate the start condition on the I^2C bus.
  - send the selected number of data bytes to the slave- microcontroller including the addressing information.
  - send a stop signal to the slave microcontroller at the end of the last byte to be transmitted on the I^2C bus.
  - determine the data transit delay on the I^2C bus by:
    - starting a timer just before the start condition (refer to section 3.1.1) on the I^2C bus,
    - sending the selected number of data bytes,
    - stopping the timer just after the stop condition (refer to section 3.1.1) on the I^2C bus, and
    - obtaining the timer value.
  - return the timer value for the selected number of data bytes to the personal computer.

- Module3- Slave-µcontroller helps to:
  - receive the selected number of data bytes from the master- microcontroller including the addressing information.
  - send an acknowledgement signal for each data byte received.

5.2. Experimental Results and Analysis

The I^2C network QoS parameters, delay and throughput, identified in chapter 4 are realized using the above experimental setup (refer to figure 5.1), since single master I^2C communication is used for the experimentation, the priority QoS parameter is not considered. The delay time in **µs** and throughput in **Kbps** are measured for two different lengths (50cm and 90.5cm) of I^2C bus and the experimental results are indicated, with the corresponding theoretical values in tables 5.1, 5.2, 5.3 and 5.4 below.

No. of Data Bytes	Measured Delay Time in µs	Theoretical Delay Time in µs
1	115.1770	111.0335
2	199.6466	169.3398
3	282.8157	227.6461
4	366.6893	285.9525
5	450.1580	344.2588
6	534.1652	402.5651
7	617.8248	460.8714
8	701.4784	519.1777

Table 5.1 **Measured and Theoretical Delay Time on 400Kbps I^2C bus at 50cm length.**

No. of Data Bytes	Measured Throughput in Kbps	Theoretical Throughput in Kbps
1	173.6466	180.1258
2	145.6523	171.2533
3	134.3633	166.9258
4	128.1740	164.3630
5	124.4015	162.6683
6	121.6853	161.4646
7	119.7757	160.5654
8	118.3216	159.8682

Table 5.2 **Measured and Theoretical Throughput on 400Kbps I^2C bus at 50cm length.**

No. of	Measured Delay	Theoretical Delay

Data Bytes	Time in µs	Time in µs
1	125.3764	111.0335
2	214.2954	169.3398
3	302.9432	227.6461
4	391.7451	285.9525
5	480.6998	344.2588
6	570.2156	402.5651
7	659.1347	460.8714
8	747.8096	519.1777

Table 5.3 **Measured and Theoretical Delay Time on 400Kbps I^2C bus at 90.5cm length**

No. of Data Bytes	Measured Throughput in Kbps	Theoretical Throughput in Kbps
1	159.1996	180.1258
2	135.3276	171.2533
3	125.4362	166.9258
4	119.9485	164.3630
5	116.4840	162.6683
6	113.9921	161.4646
7	112.2685	160.5654
8	110.9910	159.8682

Table 5.4 **Measured and Theoretical Throughput on 400Kbps I^2C bus at 90.5cm length**

As it can be observed from the experimental results and theoretical values in tables 5.1, 5.2, 5.3 and 5.4, the differences between the measured and theoretical values for the QoS parameters (delay and throughput) are less for shorter bus length. For instance at 50cm bus length the deviations of the theoretical delay from the measured one varies from 3.73% to 35.11% for one byte to eight bytes data transfers respectively, but the deviations at 90.5cm bus length varies from 12.92% to 44.04% for one byte to eight bytes data transfers respectively. Moreover, this experiment indicates that better values can be obtained for both delay and throughput at shorter bus lengths, for example at 50cm bus length the measured delay and throughput for one byte data transfer are 115.1770 µs and 173.6466Kbps respectively; whereas at 90.5cm bus length they are 125.3764 µs and 159.1996 respectively

for the same one byte data transfer. The differences between the theoretical and measured values for both delay and throughput at 50cm and 90.5cm I^2C bus lengths can be seen from the graphs indicated in figures 5.2 up to 5.5 below.

The major drawback of the theoretical formula is that it does not incorporate the effect of the I^2C bus lengths which causes larger deviations as indicated using the 90.5cm bus length instead of the 50cm bus length for the same experimental setup. I.e. the baudrate depends on the transmission distance and this effect is not incorporated into the theoretical formula. The second factor for the deviation is that the time to check whether the bus is free or not to begin transmission by the transmitter and receiver is not included in the theoretical formula. And also the frequency (7.373MHz) of the microcontroller has a tolerance of $\pm$ 2.5% [20] and this is not included in the theoretical formula. In order to come up with a theoretical formula that considers the above mentioned factors and others; experiments should be conducted using many different I^2C bus lengths. Moreover, the experimental results indicate the possibility of obtaining better delay and throughput values with small variations from the measured ones at shorter I^2C bus lengths.

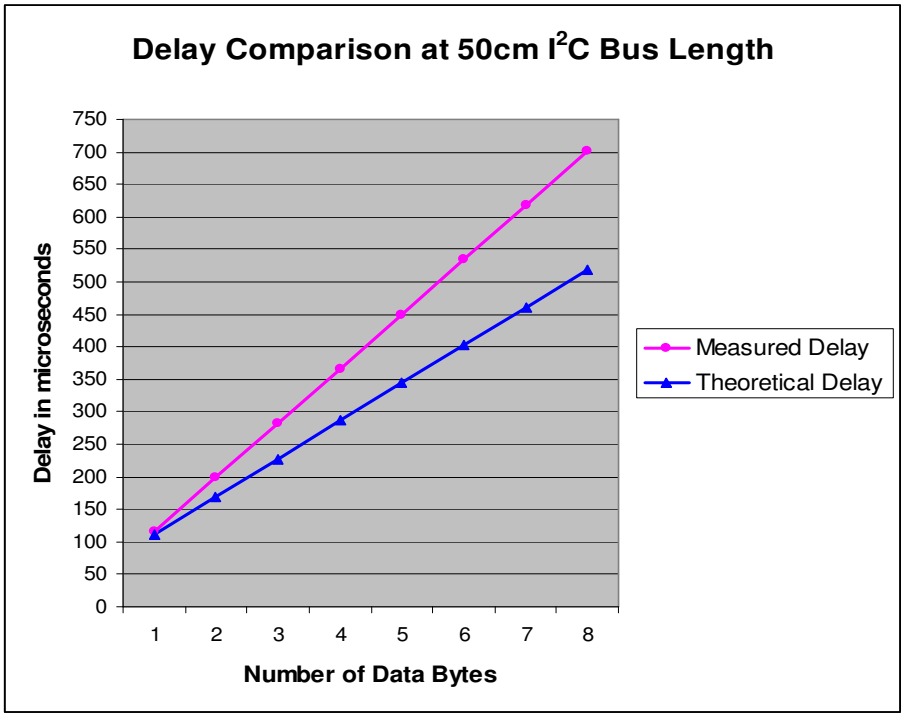


Figure 5.2 **Measured and Theoretical Delay at 50cm bus length**

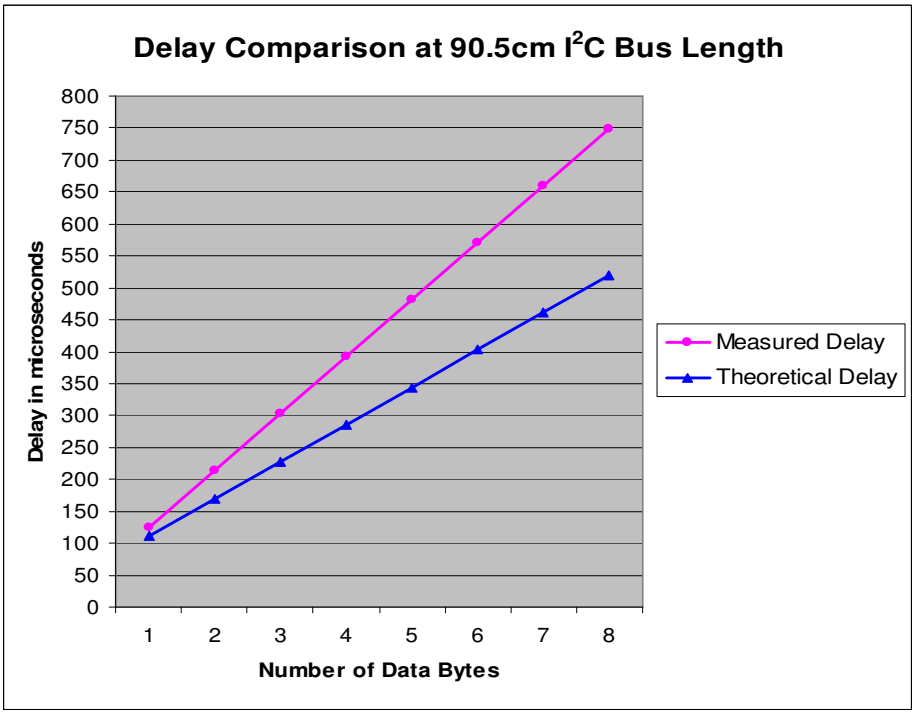


Figure 5.3 **Measured and Theoretical Delay at 90.5cm bus length**

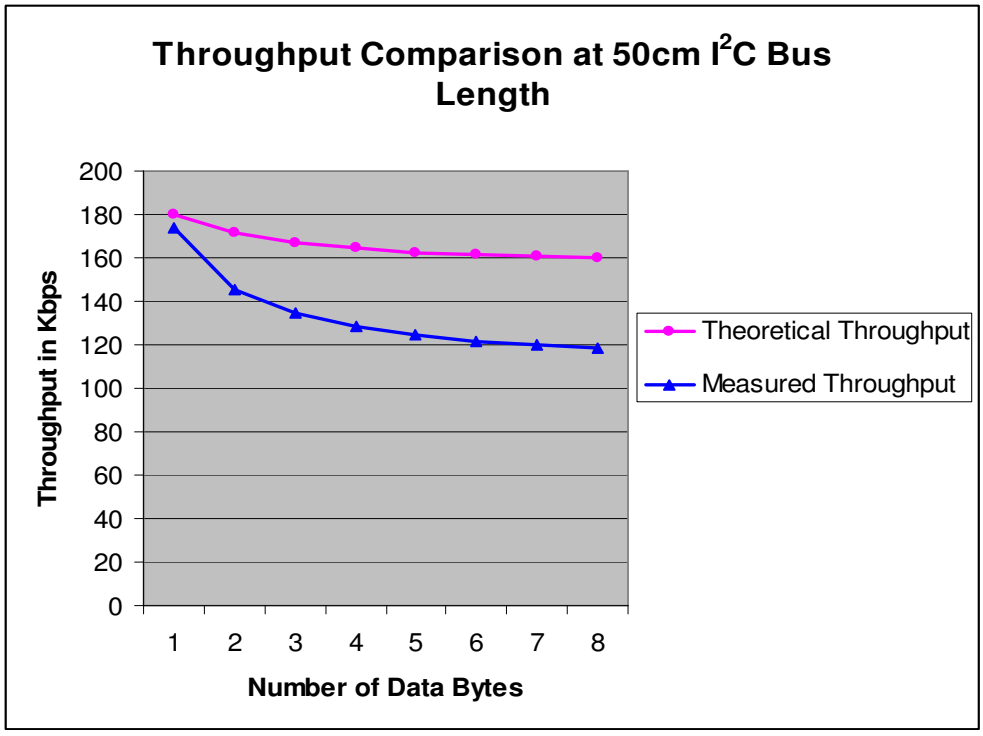


Figure 5.4 **Measured and Theoretical Throughput at 50cm bus length**

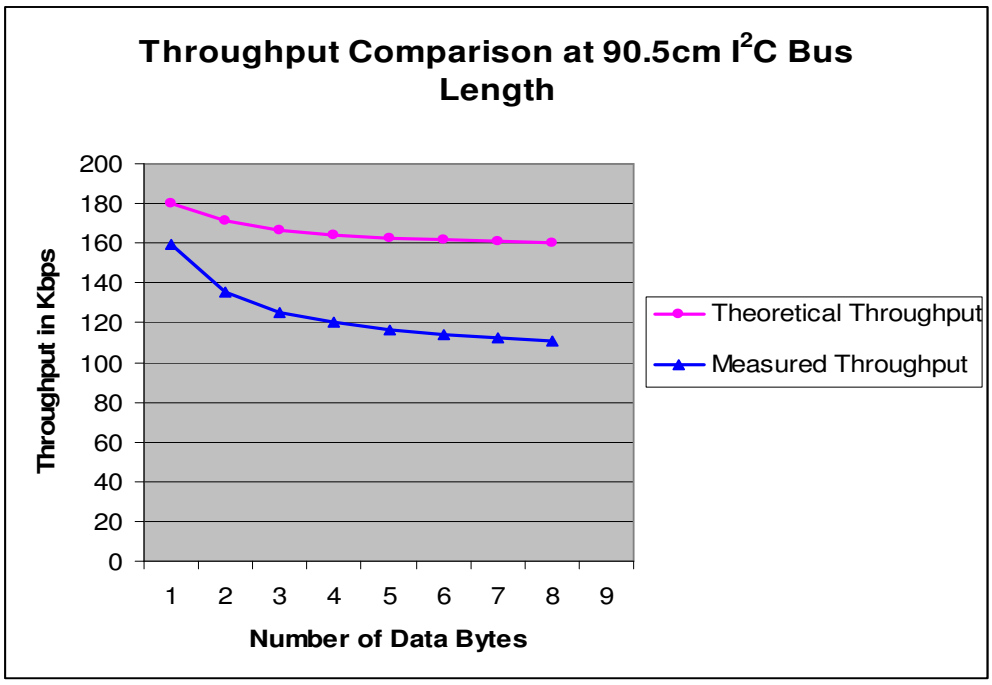


Figure 5.5 **Measured and Theoretical Throughput at 90.5cm bus length**

CHAPTER 6. Contributions and Conclusions

6.1. Contributions

The major contributions of this research work are:

- MDA approach for embedded software development by separating the concerns of development as application and network (priority based) has been proposed
- The QoS based PM specification for the communication subsystem gives QoS parameters that a given network in the priority based category can support so that an application can be developed based on the relevant QoS parameters.
- This thesis work can be used as a guide for those who will be interested to continue their research works on the embedded software development process using QoS based MDA approach and UML concepts.
- The experimental setup can be used as a guide in order to obtain better values of the supported QoS parameters at different bus lengths based on the requirements of a given application.

6.2. Conclusions

This thesis work has introduced an adaptation of the MDA toward the development of embedded software. The UML profiles are used in defining the Resource and QoS artifacts in the PIM, PM and PSM of the developed conceptual model. This model, which is developed by employing QoS based MDA approach and UML concepts, contains PIM (application QoS requirements and functional elements), PM (target networks services and supported QoS parameters), mapping layer (separates the two concerns of development as application and network) and PSM (contains information about the appropriate network on which the application will be implemented). The mapping layer uses the mapping procedure to transform inputs from the PIM and PM into a PSM that will be used for implementation.

The experiment is conducted to measure the delay and throughput of the I^2C network using different I^2C bus lengths (50cm and 90.5cm) for different number of data bytes (one to eight bytes). The main intension of this experiment is to compare the theoretical delay and throughput values that are used in the PM of the conceptual model with the corresponding measured values. The experimental results show that the differences between the measured and theoretical values for these QoS parameters are less for shorter bus length. Moreover, better values can be obtained for both delay and throughput at shorter bus lengths.

And also the experiment is used to demonstrate a means of measuring and providing delay and throughput related information of this specific network at 50cm and 90.5cm bus lengths to the mapping layer. If complete information about QoS parameters for the entire communication subsystem of the embedded platforms are given to the mapping layer, it easy for the embedded application developers to specify their QoS requirements. The mapping layer will compare these requirements with the supported values of the QoS parameters of different networks in order to select a network that is most appropriate for the application to be implemented. Hence the mapping layer separates the application specification from its implementation as two different concerns of development. Thus, if information about new network is included in the mapping layer that can best fit the existing application, then the mapping layer can select this network as the target platform without modifying the application specification.

CHAPTER 7. Future Works and Limitations

7.1. Future Works

- The effect of including the mapping layer on the embedded system devices needs further research in order to analyze the effect of this layer on the limited resources of the domain of embedded system and also its impact on the QoS parameter values of the services between the communicating applications. This can be considered as a future work.
- Designing an integrated MDA methodology with a mapping layer that can hide the details at the architectural platform level of the domain of embedded systems requires future research. If all the network communication subsystem, device driver and RTOS of an embedded system are dealt with using MDA approach, the embedded system application developers can work independently and more efficiently on an application instead of being concerned with the details of target platforms. The platform based design approach enables the developers to choose the appropriate abstraction level that can avoid their concerns about the details of the platforms.
- Experiments on the CAN network in order to obtain the values of the supported QoS parameters can be conducted in the future work.
- In order to choose different networks as the target platforms using the mapping layer, the QoS requirements for different applications (hard real-time, soft real-time, non real-time) should be studied as a future work.

7.2. Limitations

- This thesis work focused on the priority based category of the communication subsystem particularly on I^2C and CAN. The QoS based MDA approach for these networks is done as presented in chapter 4. The implementation to measure some of the supported QoS parameters as it presented in chapter 5 is done only for the I^2C network due to the unavailability of CAN network.
- The code generation is not considered in this work due to lack of full fledged mapping layer that comprises of complete libraries, tools etc. And also other elements of the architectural platform such as I/O devices, operating systems are not dealt with using the MDA approach.

- In this work, only 2 different I^2C bus lengths (50cm and 90.5cm) are used due to the unavailability of the required hardware resources in the local market.

References

1. A.S.-Vincentelli and G.Martin, “Platform-Based Design and Software Design Methodology for Embedded Systems”, IEEE Design & Test, v.18 n.6, p.23-33, November 2001.

2. A.Solberg, K.Husa, J.Aagedal and E.Abrahamsen,” QoS-aware MDA”, Paper presented at the Implementation and Validation of Object-oriented Embedded Systems (SIVOES-MDA'2003) in conjunction with UML'2003.

3. S.Deelstra, M.Sinnema, J.V.Gurp and J.Bosch, “Model Driven Architecture as Approach to Manage Variability in Software Product Families”, In Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications (MDAFA 2003), p.109-114, CTIT Technical Report TR-CTIT-03-27.

4. ARTIST. “Adaptive Real-Time Systems for Quality of Service Management”, Draft IST-2001-34820/2003

5. CAE Specification, “Data Link Provider Interface (DLPI)”, X/Open Document Number: C614, 1997.

6. M.Grimheden and M.Törngren, “What is embedded systems and how should it be taught?”, Results from a Didactic Analysis, ACM Transactions on Embedded Computing Systems (TECS) v. 4 n.3, p.633-651, August 2005 ACM Press, New York, NY, USA.

7. E.Alemu, D.Bekele and J.-P.Babau, “MDA Approach for the Development of Embeddable Applications on Communicating Devices”, MDEIS 2006: 88-97.

8. H.Kopetz “Design Principles for Distributed Embedded Applications.” Kluwer Academic Publisher, 1997.

9. ISO International Standard 11898 - Road vehicles - Interchange of digital information - Controller Area Network (CAN) for high-speed communication, November 1993.

10. J.P.Almeida, D.Dijkman, M.V.Sinderen and L.F.Pires, “Handling QoS in MDA and On the Notion of Abstract in MDA Development:” A Discussion on Availability and Dynamic Reconfiguration, CTIT Technical Report TR–CTIT–03–27, University of Twente.

11. J.D.Poole, “Model Driven Architecture Vision, Standards and Emerging Technologies”, Workshop on Metamodeling and Adaptive Object Models, 2002.

12. J.L.Houberdon and J.P.Babau, “MDA for Embedded Systems Dedicated to Process Control, SIVOEES-MDA, 2003”

13. L.Rauchaupt, “Performance Analysis of CAN based systems.” In proceedings of the first International CAN conference, Mentnz, Germany, September 1994.

14. M. H. Lee “Model Based Reasoning: A Principled Approach, Software Concepts and Tools” – 2000, Journal, p. 179-189.

15. OMG, “MDA Guide, v1.0.1”, omg/03-06-01, June 2003.

16. S.Kabanda and M.Adigun, “Extending Model Driven Architecture Benefits to Requirements Engineering”, ACM International Conference Proceeding Series, v. 204, p.22 – 30, October 2006 ACM Press, Somerset West, South Africa.

17. OMG, “Meta Object Facility(MOF)2.0 Core Specification”, ptc/03-10-04, Oct. 2003.

18. Model-Driven Architecture - A Technical Perspective, Technical Report ORMSC/2001-07-01, Framingham, MA: Object Management Group, July 2001.

19. OMG, “UML Profile for Schedualbility, Performance, and Time Specification”, v1.0 formal /03-09-01, 2003.

20. P89LPC932 8-bit Microcontroller with Accelerated two-clock 80C51 Core 8KB Flash with 512-byte Data EEPROM and 768-byte RAM, Rev.04-06 January 2004.

21. P.Boulet, J.L.Dekeyser, C.Dumoulin and P.Marquet, "MDA for SoC Embedded Systems Design, Intensive Signal Processing Experiment", SIVOES-MDA Workshop at UML 2003, p. 20-24 October 2003 San Francisco.

22. Request for Proposal: MOF 2.0 Query/Views/Transformations RFP. Technical Report ad/2002-04-10, Object Management Group, 2002.

23. Robert Bosch, “CAN Specification”, Version 2.0, 1991.

24. R.Chen, M.Sgroi, L.Lavagno, G.Martin, A.S.-Vincentelli and J.Rabaey, “UML and Platform-based Design, UML for Real: Design of Embedded Real-time Systems”, Kluwer Academic Publishers, Norwell, MA, 2003.

25. The I^2C Specification, Version 2.1, January 2000:

26. Tournier, J.-Charles and J.-P.Babau, “Communication Protocol Evaluation for Embedded Systems”, Industrial Technology, 2003 IEEE International Conference, v.2.

27. OMG, “UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms”, OMG Adopted Specification 2004.

28. OMG, “Unified Modeling Language(UML) Specification: Infrastructure”, version 2.0, ptc/03-09-15, Sept. 2003.

29. W.Stallings, “Data and Computer Communications”, Fifth Edition, Printice Hall, 2000.

30. E. Riccobene, P. Scandurra, A. Rosti, S. Bocchio, “A UML 2.0 Profile for SystemC: Toward High-level SoC Design”, In Proceedings of the 5th ACM International Conference on Embedded Software, September 18-22, 2005, Jersey City, NJ, USA.

31. M.Edwards and P.Green, “UML for Hardware and Software Object Modeling, UML for Real: Design of Embedded Real-time Systems”, Kluwer Academic Publishers, Norwell, MA, 2003.

32. T.Kangas, P.Kukkala, H.Orsila, E.Salminen, M.Hännikäinen, T.D. Hämäläinen, J.Riihimäki and K.Kuusilinna, “UML-based Multiprocessor SoC Design Framework”, ACM Transactions on Embedded Computing Systems (TECS), v.5 n.2, p.281-320, May 2006.

33. L.Sha, “Open Challenges in Real-time Embedded Systems”, ACM SIGBED Review, v.1n.1, p.13-15, April 2004 ACM Press, New York, NY, USA.

34. G.Buttazzo, “Research Trends in Real-time Computing for Embedded Systems”, Special Issue on Major International Initiatives on Real-time and Embedded Systems, v.3 n.3, p1-10, July 2006 ACM Press, New York, NY, USA.

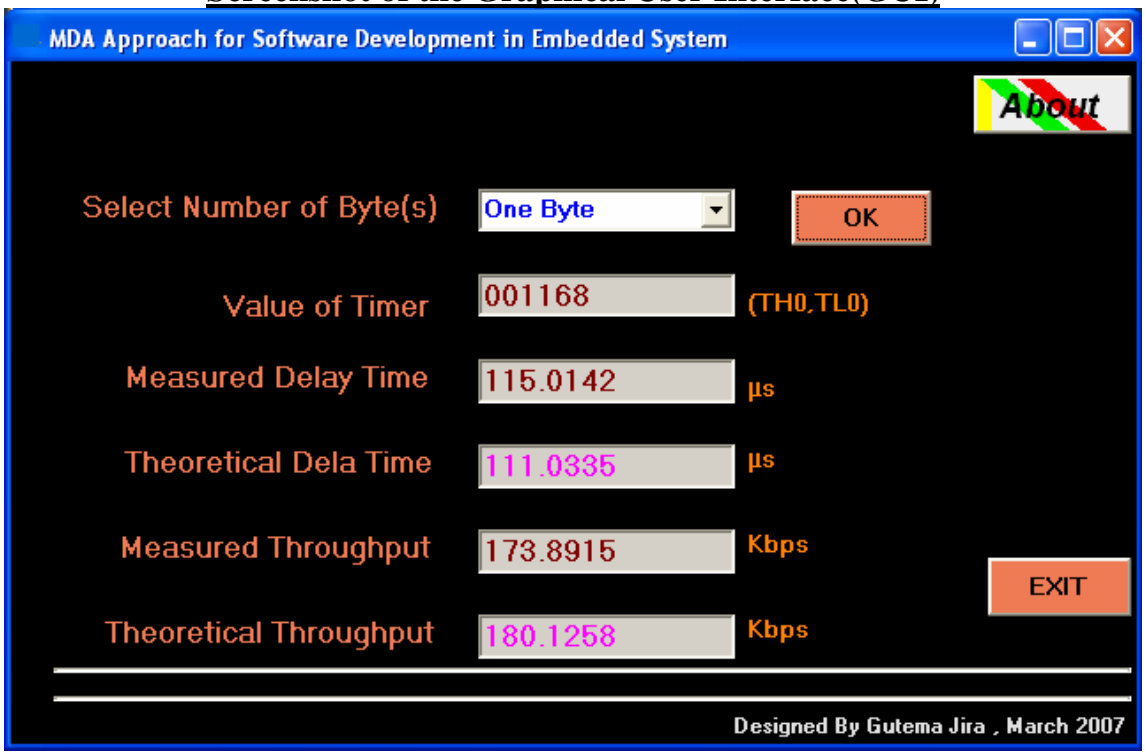
35. G.Caplat and J.-L.Sourouille, “Model Mapping in MDA”, In Workshop wiSME UML’2002, Dresden, Germany, 2002.

36. Z.Gu, S.Kodase, S.Wang and K.G.Shih, “A Model-based Approach to System-level Dependency and Real-time Analysis of Embedded Software”, In Proceeding of the Real-time and Embedded Technology and Application Symposium RTAS’03, Toronto, Canada, May 2003, IEEE.

37. E.Cheong, J.Liebman, J.Liu and F.Z.Tinygals, “A programming Model for Event-Driven Embedded Systems”, In Proceedings of the 2003 ACM Symposium on Applied Computing, p.698-704, ACM press, 2003.

ANNEX A

Screenshot of the Graphical User Interface(GUI)



ANNEX B

Schematic Circuit Diagram - I^2C Master Side (Next Page)

Schematic Circuit Diagram – I^2C Slave Side (Next Page)

ANNEX C

80C51 FAMILY INSTRUCTION SET

MNEMONIC		DESCRIPTION	BYTE	OSCILLATOR PERIOD
ARITHMETIC OPERATIONS				
ADD	A,Rn	Add register to Accumulator	1	12
ADD	A,direct	Add direct byte to Accumulator	2	12
ADD	A,@Ri	Add indirect RAM to Accumulator	1	12
ADD	A,#data	Add immediate data to Accumulator	2	12
ADDC	A,Rn	Add register to Accumulator with carry	1	12
ADDC	A,direct	Add direct byte to Accumulator with carry	2	12
ADDC	A,@Ri	Add indirect RAM to Accumulator with carry	1	12
ADDC	A,#data	Add immediate data to A_{CC} with carry	2	12
SUBB	A,Rn	Subtract Register from A_{CC} with borrow	1	12
SUBB	A,direct	Subtract direct byte from A_{CC} with borrow	2	12
SUBB	A,@Ri	Subtract indirect RAM from A_{CC} with borrow	1	12
SUBB	A,#data	Subtract immediate data from A_{CC} with borrow	2	12
INC	A	Increment Accumulator	1	12
INC	Rn	Increment register	1	12

80C51 FAMILY INSTRUCTION SET(Continued)

MNEMONIC		DESCRIPTION	BYTE	OSCILLATOR PERIOD
ARITHMETIC OPERATIONS (Continued)				
INC	direct	Increment direct byte	2	12
INC	@Ri	Increment indirect RAM	1	12
DEC	A	Decrement Accumulator	1	12
DEC	Rn	Decrement Register	1	12
DEC	direct	Decrement direct byte	2	12
DEC	@Ri	Decrement indirect RAM	1	12
INC	DPTR	Increment Data Pointer	1	24
MUL	AB	Multiply A and B	1	48
DIV	AB	Divide A by B	1	48
DA	A	Decimal Adjust Accumulator	1	12
LOGICAL OPERATIONS				
ANL	A,Rn	AND Register to Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@Ri	AND indirect RAM to Accumulator	1	12
ANL	A,#data	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct,#data	AND immediate data to direct byte	3	24
ORL	A,Rn	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12
ORL	A,@Ri	OR indirect RAM to Accumulator	1	12
ORL	A,#data	OR immediate data to Accumulator	2	12
ORL	direct,A	OR Accumulator to direct byte	2	12
ORL	direct,#data	OR immediate data to direct byte	3	24
XRL	A,Rn	Exclusive-OR register to Accumulator	1	12
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	12
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	12
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	12
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3	24
CLR	A	Clear Accumulator	1	12
CPL	A	Complement Accumulator	1	12
RL	A	Rotate Accumulator left	1	12
RLC	A	Rotate Accumulator left through the carry	1	12
RR	A	Rotate Accumulator right	1	12
RRC	A	Rotate Accumulator right through the carry	1	12
SWAP	A	Swap nibbles within the Accumulator	1	12
DATA TRANSFER				
MOV	A,Rn	Move register to Accumulator	1	12
MOV	A,direct	Move direct byte to Accumulator	2	12
MOV	A,@Ri	Move indirect RAM to Accumulator	1	12

80C51 FAMILY INSTRUCTION SET(Continued)

MNEMONIC		DESCRIPTION	BYTE	OSCILLATOR PERIOD
DATA TRANSFER (Continued)				
MOV	A,#data	Move immediate data to Accumulator	2	12
MOV	Rn,A	Move Accumulator to register	1	12
MOV	Rn,direct	Move direct byte to register	2	24
MOV	RN,#data	Move immediate data to register	2	12
MOV	direct,A	Move Accumulator to direct byte	2	12
MOV	direct,Rn	Move register to direct byte	2	24
MOV	direct,direct	Move direct byte to direct	3	24
MOV	direct,@Ri	Move indirect RAM to direct byte	2	24
MOV	direct,#data	Move immediate data to direct byte	3	24
MOV	@Ri,A	Move Accumulator to indirect RAM	1	12
MOV	@Ri,direct	Move direct byte to indirect RAM	2	24
MOV	@Ri,#data	Move immediate data to indirect RAM	2	12
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	24
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A_{CC}	1	24
MOVC	A,@A+PC	Move Code byte relative to PC to A_{CC}	1	24
MOVX	A,@Ri	Move external RAM (8-bit addr) to A_{CC}	1	24
MOVX	A,@DPTR	Move external RAM (16-bit addr) to A_{CC}	1	24
MOVX	A,@Ri,A	Move A_{CC} to external RAM (8-bit addr)	1	24
MOVX	@DPTR,A	Move A_{CC} to external RAM (16-bit addr)	1	24
PUSH	direct	Push direct byte onto stack	2	24
XCH	A,Rn	Exchange register with Accumulator	1	12
XCH	A,direct	Exchange direct byte with Accumulator	2	12
XCH	A,@Ri	Exchange indirect RAM with Accumulator	1	12
XCHD	A,@Ri	Exchange low-order digit indirect RAM with A_{CC}	1	12
BOOLEAN VARIABLE MANIPULATION				
CLR	C	Clear carry	1	12
CLR	bit	Clear direct bit	2	12
SETB	C	Set carry	1	12
SETB	bit	Set direct bit	2	12
CPL	C	Complement carry	1	12
CPL	bit	Complement direct bit	2	12
ANL	C,bit	AND direct bit to carry	2	24
ANL	C,/bit	AND complement of direct bit to carry	2	24
ORL	C,bit	OR direct bit to carry	2	24
ORL	C,/bit	OR complement of direct bit to carry	2	24
MOV	C,bit	Move direct bit to carry	2	12
MOV	bit,C	Move carry to direct bit	2	24
JC	rel	Jump if carry is set	2	24
JNC	rel	Jump if carry not set	2	24

80C51 FAMILY INSTRUCTION SET(Continued)

MNEMONIC		DESCRIPTION	BYTE	OSCILLATOR PERIOD
BOOLEAN VARIABLE MANIPULATION (Continued)				
JB	rel	Jump if direct bit is set	3	24
JNB	rel	Jump if direct bit is not set	3	24
JBC	bit,rel	Jump if direct bit is set and clear bit	3	24
PROGRAM BRANCHING				
ACALL	addr11	Absolute subroutine call	2	24
LCALL	addr16	Long subroutine call	3	24
RET		Return from subroutine	1	24
RETI		Return from interrupt	1	24
AJMP	addr11	Absolute jump	2	24
LJMP	addr16	Long jump	3	24
SJMP	rel	Short jump (relative addr)	2	24
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	24
JZ	rel	Jump if Accumulator is zero	2	24
JNZ	rel	Jump if Accumulator is not zero	2	24
CJNE	A,direct,rel	Compare direct byte to A_{CC} and jump if not equal	3	24
CJNE	A,#data,rel	Compare immediate to A_{CC} and jump if not equal	3	24
CJNE	RN,#data,rel	Compare immediate to register and jump if not equal	3	24
CJNE	@Ri,#data,rel	Compare immediate to indirect and jump if not equal	3	24
DJNZ	Rn,rel	Decrement register and jump if not zero	2	24
DJNZ	direct,rel	Decrement direct byte and jump if not zero	3	24
NOP		No operation	1	12

ANNEX D

Assembly Code

```
C51 COMPILER V8.08   I2C_MASTER

ASSEMBLY LISTING OF GENERATED OBJECT CODE

             ; FUNCTION _i2c_transmit (BEGIN)

                          . . .

000F 120000      R     LCALL   timers_starttimer0
0012 D2DD              SETB    STA
0014 7F02              MOV     R7,#02H
0016 22                RET
             ; FUNCTION _i2c_transmit (END)

             ; FUNCTION i2c_getstatus (BEGIN)

0000 20DBFD            JB      SI,?C0019
0003 AF00        R     MOV     R7,mi2cstatus
0005 22                RET
             ; FUNCTION i2c_getstatus (END)

             ; FUNCTION _i2c_master_getbyte (BEGIN)
0000 7400        R     MOV     A,#LOW I2C_DATA_TX
0002 2F                ADD     A,R7
0003 F8                MOV     R0,A
0004 E6                MOV     A,@R0
0005 FF                MOV     R7,A
0006 22                RET
             ; FUNCTION _i2c_master_getbyte (END)

             ; FUNCTION _i2c_master_islasttxbyte (BEGIN)
0000 7400        R     MOV     A,#LOW I2C_DATA_TX
0002 2F                ADD     A,R7
0003 F8                MOV     R0,A
0004 E6                MOV     A,@R0
0005 B42A03            CJNE    A,#02AH,?C0023
0008 7F01              MOV     R7,#01H
000A 22                RET
000B 7F00              MOV     R7,#00H
000D 22                RET
             ; FUNCTION _i2c_master_islasttxbyte (END)

             ; FUNCTION i2c_isr (BEGIN)
0000 C0E0              PUSH    ACC
0002 C0D0              PUSH    PSW
0004 75D008            MOV     PSW,#08H
0007 E5D9              MOV     A,I2STAT
0009 54F8              ANL     A,#0F8H
000B FF                MOV     R7,A
000C 24F0              ADD     A,#0F0H
000E 6018              JZ      ?C0028
0010 24F8              ADD     A,#0F8H
0012 6023              JZ      ?C0029
0014 24F8              ADD     A,#0F8H
0016 6023              JZ      ?C0030
0018 24F8              ADD     A,#0F8H
001A 6021              JZ      ?C0031
001C 24F8              ADD     A,#0F8H
001E 601B              JZ      ?C0030
0020 24F8              ADD     A,#0F8H
0022 604E              JZ      ?C0035
0024 2430              ADD     A,#030H
0026 7050              JNZ     ?C0036
0028 8500DA      R     MOV     I2DAT,mslaveaddress
002B C2DD              CLR     STA
002D C2DC              CLR     STO
```

```
002F 750000      R     MOV     mbytenum,#00H
0032 750000      R     MOV     mbytenum+01H,#00H
0035 804A              SJMP    ?C0026
0037 AF00        R     MOV     R7,mbytenum+01H
0039 8021              SJMP    ?C0078
003B 802C              SJMP    ?C0079
003D AF00        R     MOV     R7,mbytenum+01H
003F AE00        R     MOV     R6,mbytenum
0041 120000      R     LCALL   _i2c_master_islasttxbyte
0044 EF                MOV     A,R7
0045 600C              JZ      ?C0032
0047 C2DD              CLR     STA
0049 D2DC              SETB    STO
004B 120000      R     LCALL   timers_stoptimer0
004E 750002      R     MOV     mi2cstatus,#02H
0051 802E              SJMP    ?C0026
0053 0500        R     INC     mbytenum+01H
0055 E500        R     MOV     A,mbytenum+01H
0057 7002              JNZ     ?C0075
0059 0500        R     INC     mbytenum
005B FF                MOV     R7,A
005C AE00        R     MOV     R6,mbytenum
005E 120000      R     LCALL   _i2c_master_getbyte
0061 8FDA              MOV     I2DAT,R7
0063 C2DD              CLR     STA
0065 C2DC              CLR     STO
0067 8018              SJMP    ?C0026
0069 C2DD              CLR     STA
006B D2DC              SETB    STO
006D 750004      R     MOV     mi2cstatus,#04H
0070 800F              SJMP    ?C0026
0072 D2DD              SETB    STA
0074 C2DC              CLR     STO
0076 8009              SJMP    ?C0026
0078 750004      R     MOV     mi2cstatus,#04H
007B C2DD              CLR     STA
007D C2DC              CLR     STO
007F D2DA              SETB    AA
0081 C2DB              CLR     SI
0083 D0D0              POP     PSW
0085 D0E0              POP     ACC
0087 32                RETI
               ; FUNCTION i2c_isr (END)

               ; FUNCTION TransmitData (BEGIN)
0000 7F02              MOV     R7,#02H
0002 120000        R   LCALL   _i2c_transmit
0005 120000        R   LCALL   i2c_getstatus
0008 EF                MOV     A,R7
0009 20E0F9            JB      ACC.0,?C0038
000C 22                RET
               ; FUNCTION TransmitData (END)
```

```
C51 COMPILER V8.08   SLAVE

ASSEMBLY LISTING OF GENERATED OBJECT CODE


               ; FUNCTION _i2c_slave_receivedbyte (BEGIN)
0000 7400          R   MOV     A,#LOW I2C_DATA_RX
0002 2F                ADD     A,R7
0003 F8                MOV     R0,A
0004 A605              MOV     @R0,AR5
0006 22                RET
               ; FUNCTION _i2c_slave_receivedbyte (END)
```

```
                        ; FUNCTION _i2c_slave_islastrxbyte (BEGIN)
0000 7400                  R     MOV     A,#LOW I2C_DATA_RX
0002 2F                          ADD     A,R7
0003 F8                          MOV     R0,A
0004 E6                          MOV     A,@R0
0005 B42A05                      CJNE    A,#02AH,?C0007
0008 D200                  R     SETB    START
000A 7F01                        MOV     R7,#01H
000C 22                          RET
000D 7F00                        MOV     R7,#00H
000F 22                          RET
                        ; FUNCTION _i2c_slave_islastrxbyte (END)

                        ; FUNCTION i2c_isr (BEGIN)
0000 C0E0                        PUSH    ACC
0002 C083                        PUSH    DPH
0004 C082                        PUSH    DPL
0006 C0D0                        PUSH    PSW
0008 75D008                      MOV     PSW,#08H
000B E5D9                        MOV     A,I2STAT
000D 54F8                        ANL     A,#0F8H
000F FF                          MOV     R7,A
0010 120000                E     LCALL   ?C?CCASE
0013 0000                  R     DW      ?C0012
0015 08                          DB      08H
0016 0000                  R     DW      ?C0012
0018 10                          DB      010H
0019 0000                  R     DW      ?C0016
001B 60                          DB      060H
001C 0000                  R     DW      ?C0016
001E 68                          DB      068H
001F 0000                  R     DW      ?C0016
0021 70                          DB      070H
0022 0000                  R     DW      ?C0016
0024 78                          DB      078H
0025 0000                  R     DW      ?C0020
0027 80                          DB      080H
0028 0000                  R     DW      ?C0024
002A 88                          DB      088H
002B 0000                  R     DW      ?C0020
002D 90                          DB      090H
002E 0000                  R     DW      ?C0024
0030 98                          DB      098H
0031 0000                  R     DW      ?C0025
0033 A0                          DB      0A0H
0034 0000                        DW      00H
0036 0000                  R     DW      ?C0026
0038 8500DA                R     MOV     I2DAT,mslaveaddress
003B C2DD                        CLR     STA
003D C2DC                        CLR     STO
003F 750000                R     MOV     mbytenum,#00H
0042 750000                R     MOV     mbytenum+01H,#00H
0045 806E                        SJMP    ?C0010
0047 C2DD                        CLR     STA
0049 C2DC                        CLR     STO
004B 750000                R     MOV     mbytenum,#00H
004E 750000                R     MOV     mbytenum+01H,#00H
0051 AF00                  R     MOV     R7,mbytenum+01H
0053 AE00                  R     MOV     R6,mbytenum
0055 120000                R     LCALL   _i2c_slave_islastrxbyte
0058 EF                          MOV     A,R7
0059 6004                        JZ      ?C0017
005B C2DA                        CLR     AA
005D 8056                        SJMP    ?C0010
005F D2DA                        SETB    AA
0061 8052                        SJMP    ?C0010
0063 ADDA                        MOV     R5,I2DAT
0065 AF00                  R     MOV     R7,mbytenum+01H
0067 AE00                  R     MOV     R6,mbytenum
0069 120000                R     LCALL   _i2c_slave_receivedbyte
006C 0500                  R     INC     mbytenum+01H
006E E500                  R     MOV     A,mbytenum+01H
0070 7002                        JNZ     ?C0040
```

```
0072 0500        R     INC     mbytenum
0074 C2DD              CLR     STA
0076 C2DC              CLR     STO
0078 FF                MOV     R7,A
0079 AE00        R     MOV     R6,mbytenum
007B 120000      R     LCALL   _i2c_slave_islastrxbyte
007E EF                MOV     A,R7
007F 6004              JZ      ?C0021
0081 C2DA              CLR     AA
0083 8030              SJMP    ?C0010
0085 D2DA              SETB    AA
0087 802C              SJMP    ?C0010
0089 ADDA              MOV     R5,I2DAT
008B AF00        R     MOV     R7,mbytenum+01H
008D AE00        R     MOV     R6,mbytenum
008F 120000      R     LCALL   _i2c_slave_receivedbyte
0092 0500        R     INC     mbytenum+01H
0094 E500        R     MOV     A,mbytenum+01H
0096 7002              JNZ     ?C0041
0098 0500        R     INC     mbytenum
009A 750002      R     MOV     mi2cstatus,#02H
009D 8010              SJMP    ?C0043
009F 750002      R     MOV     mi2cstatus,#02H
00A2 C2DD              CLR     STA
00A4 C2DC              CLR     STO
00A6 D2DA              SETB    AA
00A8 D200        R     SETB    START
00AA 8009              SJMP    ?C0010
00AC 750004      R     MOV     mi2cstatus,#04H
00AF C2DD              CLR     STA
00B1 C2DC              CLR     STO
00B3 D2DA              SETB    AA
00B5 C2DB              CLR     SI
00B7 D0D0              POP     PSW
00B9 D082              POP     DPL
00BB D083              POP     DPH
00BD D0E0              POP     ACC
00BF 32                RETI
             ; FUNCTION i2c_isr (END)
```

To transmit the first data byte including the addressing and control information, the master executes the following number of instructions.

No.	Name	No. of Repeats	#instructi
1	i2c_getstatus	3	6
2	i2c_master_islasttxbyte	1	12
3	i2c_master_getbyte	1	7
4	i2c_isr	3	6
5	slave address transmit	1	40
6	first data byte transmit	1	9
7	stop condition	1	8
Total_1			

To receive the first data byte including the addressing and control information, the slave executes the following number of instructions.

No.	Name	No. of Repeats	#instruct
1	i2c_slave_receivedbyte	3	7
2	i2c_slave_islastrxbyte	1	15
3	i2c_isr	2	10
4	slave address receive	1	26
5	first data byte receive	1	21
6	stop condition	1	10
Total_2			

To transmit one more data byte, the master executes the following number of instructions.

No.	Name	No. of Repeats	#instruct
1	i2c_getstatus	1	6
2	i2c_master_islasttxbyte	1	7
3	i2c_master_getbyte	1	12
4	i2c_isr	2	6
5	one more data byte transmit	1	23
Total_3			

To receive one more data byte, the slave executes the following number of instructions.

No.	Name	No. of Repeats	#instruct
1	i2c_slave_receivedbyte	1	7
2	i2c_islastrxbyte	1	15
3	i2c_isr	2	10
4	one more data byte receive	1	50
Total_4			