App Your Car

By

Ahmed Hisham Saad

Mostafa Mohamed Kamel

Mahmoud Sameh Mahmoud

Yasser Mohamed Ramadan

A Thesis submitted to the

Faculty of Engineering at Cairo University

in partial fulfilment of the requirments

for the Degree of Bachelor of Science in

ELECTRONICS AND COMMUNICATIONS ENGINEERING

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2015

App Your Car

By

Ahmed Hisham Saad

Mostafa Mohamed Kamel

Mahmoud Sameh Mahmoud

Yasser Mohamed Ramadan

A Thesis submitted to the

Faculty of Engineering at Cairo University

in partial fulfilment of the requirments

for the Degree of Bachelor of Science in

ELECTRONICS AND COMMUNICATIONS ENGINEERING

Under the Supervision of

Dr. Hassan Mostafa

Assistant Professor

Electronics and Communications Department

Faculty of Engineering, Cairo University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2015

"This page intentionally left blank"

Table of Contents

Table of Contents .... **I**
List of Figures .... **IV**
List of Tables .... **VIII**
Nomenclature .... **IX**
Acknowledgements .... **X**
Abstract .... **XI**
1 Introduction **1**
1.1 Evolution in vehicle 1
1.2 Evolution in Mobile 1
1.3 Thesis Organization and Overview of Methods and Contributions 2
1.3.1 Chapter 2: Motivation 2
1.3.2 Chapter 3: Overview & Background 2
1.3.3 Chapter 4: Project Description & Related Work 3
1.3.4 Chapter 5: Technical Approach 3
1.3.5 Chapter 6: Implementation and Results 3
1.3.6 Chapter 7: Conclusions and Future Work 4
2 Motivation **5**
2.1 Internet of Things 5
2.2 Automotive Ethernet 5
2.3 Worlds will Collide 6
3 Overview & Background **7**
3.1 Introduction 7
3.2 Automotive main building units 7
3.2.1 Electronic Control Unit (ECU) 7
3.2.2 Controller Area Network (CAN) 11
3.2.3 On Board Diagnostics 21
3.3 Embedded Web Server 27
3.3.1 Introduction 27

3.3.2 Web Servers **28**
3.3.3 The Elements of a Network **29**
3.3.4 Protocols **30**
3.3.5 Summary **36**
3.4 Mobile application **38**
3.4.1 Smartphones **38**
3.4.2 Mobile Platforms **38**
3.4.3 OS Market Share **39**
4 Project Description & Related Work **41**
4.1 Related Work **41**
4.1.1 Automotive Mobile Platform **41**
4.1.2 Auto Performance Analyzer **42**
4.2 Project Definition **42**
4.3 Project Flow Description **42**
4.4 How to choose a microcontroller **43**
4.4.1 CPU architectures **43**
4.4.2 Microcontrollers **43**
4.4.3 AVR **44**
4.4.4 Arduino **45**
4.4.5 Parameters of selecting a microcontroller **45**
4.5 Components used **46**
4.5.1 Arduino Uno **46**
4.5.2 Spark Fun CAN bus Shield **47**
4.5.3 OBD-II to DB9 Cable **48**
4.5.4 Arduino Ethernet Shield **48**
4.5.5 Portable Battery Powered 3G/3.75G Wireless/Ethernet Router **49**
4.6 Tasks Description **50**
4.6.1 Arduino Ethernet web server **50**
4.6.2 Reading real time values from different ECUs in the vehicle **50**
4.6.3 Reading the DTCs of the vehicle **50**
4.6.4 Mobile application **50**
5 Technical Approach **51**
5.1 Arduino Ethernet Web server **51**
5.2 Reading real time data from different ECUs in the vehicle **53**

5.2.1 Supported Parameters……………………………………………………**54**

5.2.2 Reading the alues of the supported Parameters……………………………..**55**

5.3 Reading the DTCs of the vehicle ………………………………………………...**57**

5.4 Mobile Application……………………………………………………………**58**

Introduction………………..………………………………………………**58**

Classes Used in the Application **60**

Some Features in App Your Car **61**

6 Implementation & Results **62**

6.1 Problems we faced and how we solve it……………………..………………...**62**

6.1.1 SPI confliction……………………………………………………………**62**

6.1.2 Delay problem……………………………………………………………**64**

6.1.3 Memory Problem………………………………………………………**64**

6.2 Adding the 4 blocks of codes together ……………………………………….**64**

6.3 What is JSON? Why we are going to use it? 64

6.4 Android Asynchronous Http Client library... **65**

6.5 SpeedometerGauge component…………. **66**

7 Conclusions and Future Work.. **67**

7.1 Conclusions…………………………………………………………………..**67**

7.2 Future Work………….. .... **67**

7.2.1 Activating the control option **67**

7.2.2 Add vehicle tracking and position information **68**

7.2.3 Support iOS Mobile Operating Systems.. **68**

7.2.4 Using the SIM card as a secure element in Android.................................... **68**

APPENDIX I **69**

PIDs and its description (Mode 1).. **69**

APPENDIX II.. **74**

Supported Parameters….. **74**

Trouble Codes…………….. **75**

Full Code………………………………………………………………………………...**76**

APPENDIX III **82**

Budget………………………………………………………………………………..…**82**

References………………………………………………………………………………...**83**

List of Figures

Figure 1 - The building blocks of our project 3
Figure 2 - in the past, automotive and ethernet technologies are separate, but now the are convergend togethe 6
Figure 3 - As of 2014, each year approximately three times more CAN nodes ship worldwide than wired Ethernet ports, an amount roughly equal to the combined total of wired Ethernet and wireless Wi-Fi controllers. 6
Figure 4 - A photo represents the huge number of ECUs inside a vehicle nowadays 8
Figure 5 - Cycle of CAN bus 8
Figure 6 - V-Model 9
Figure 7 - AUTOSAR software architecture 10
Figure 8 - The diagram on the left represents how the car ECUs were connected before using CAN bus, and the right connection is after using the CAN bus 11
Figure 9 - CAN Bus 12
Figure 10 - CAN bus and OSI model 13
Figure 11 - Encoding of dominant and recessive bits 13
Figure 12 - CAN standard Arbitration Field 14
Figure 13 - CAN Ectended Arbitration Field 14
Figure 14 - Structure of the Data Frame 15
Figure 15 - Data Frame (Start of Frame) 15
Figure 16 - Data Frame (Arbitration Field) 16
Figure 17 - Data Frame (Control Field) 16
Figure 18 - Data Frame (Data Field) 17
Figure 19 - Data Frame (CRC Field) 17
Figure 20 - Data Frame (ACK Field) 18
Figure 21 - Data Frame (End of Frame) 18
Figure 22 - Structure of the Remote Frame 19
Figure 23 - Error Frame 20
Figure 24 - Structure of the Overload Frame 20
Figure 25 - Structure of the Interframe Space 21
Figure 26 - OBD connector 21
Figure 27 - OBD connector and Pinout 22
Figure 28 - Embedded Web applications take the Internet to places previously unheard of – even a tank. 27
Figure 29 - Monitoring Home equipments in an inteligent home through Internet 28

Figure 30 - Example of a Network layered model which supports Ethernet Communications....... 29
Figure 31 - Application layer in the Network protocol stack.. 30
Figure 32 - URL beginning with the HTTP scheme and the WWW domain name label............... 30
Figure 33 - 404 error on German Wikipedia.. 31
Figure 34 - TCP & UDP in the Network protocol stack 33
Figure 35 - TCP add a header over the data 34
Figure 36 - TCP Header Format 34
Figure 37 - UDP add a header over the data.. 34
Figure 38 - UDP Header Format 35
Figure 39 - IP in the Network protocol stack 35
Figure 40 - IP address format 35
Figure 41 - A GET request simply results in a static file being returned.. 36
Figure 42 - POST requests must be handed off to some other utility that will determine an appropriate response. 37
Figure 43 - An environment is needed to allow the web server to hand off to a high-level script, which accesses low-level C routines 37
Figure 44 - Android Logo.. 38
Figure 45 - Apple Logo 39
Figure 46 - Windows Phone Logo... 39
Figure 47 - Blackberry Logo 39
Figure 48 - Mobile market share within last 4 years according to International Data Corporation (IDC), May 2015... 39
Figure 49 - Automotive Mobile Platform Ein Shams team Graduation project.............................. 41
Figure 50 - Texas university graduation project.. 42
Figure 51 - Architecture of AVR microcontroller... 44
Figure 52 - AVR.. 44
Figure 53 – Arduino Logo... 45
Figure 54 - Arduino Uno R3 Back 46
Figure 55 - Arduino Uno R3 Front.. 46
Figure 56 - Spark Fun CAN bus sheild 47
Figure 57 - OBD-II to DB9 Cable 48
Figure 58 - Ethernet Shield Front.. 48
Figure 59 - Ethernet Shield Back 48
Figure 60 - TP Link Portable Battery Powered 3G/3.75G Wireless/Ethernet Router 49
Figure 61 - The representation of the tasks of our project in an origami pyramid.......................... 50
Figure 62 - Arduino Uno R3 Front**01**... 50
Figure 63 - The HTML web page which is viewed in the browser... 53
Figure 64 - CAN bus shield attached to Arduino Uno board... 53
Figure 65 - Supported Parameters 55
Figure 66 - Serial monitor of Engine coolant temperature and engine RPM.................................. 56
Figure 67 - Displaying the values of engine RPM, distance travelled, vehicle speed, and engine coolant temperature of a stable car 56

Figure 68 - Displaying the values of engine RPM, distance travelled, vehicle speed, and engine coolant temperature of a moving car 57
Figure 69 - Flow chart of the Android Application 59
Figure 70 - The Classes that are used in the application 60
Figure 71 - changing the CS pin from D10 to D9 , Ethernet is Slave Device #1 and CAN shield is Slave Device #2 62
Figure 72 - The flow chart of the code 62
Figure 73 - The flow chart of the whole code 63
Figure 74 - The main building blocks of the project 64
Figure 75 - JSON structre format 65
Figure 76 - a simple needle gauge that looks like speedometer 66
Figure 77 - App Your Car User Interface 66

List of Tables

Table 1 - List of abbreviations and acronyms IX
Table 2 - Roles of each frame in Message frames 14
Table 3 - Discreption of the frames in Data Frame 15
Table 4 - Data Length Code DLC of Control Frame 17
Table 5 - The frames of the Remote Frame 19
Table 6 - All available PID modes (the two orange shaded modes are the most important modes)22
Table 7 - Mode 1 PIDs 23
Table 8 - Fuel Type response 23
Table 9 - Mode 2 message formation 23
Table 10 - First DTC character decoding table 24
Table 11 - Second DTC character decoding table 24
Table 12 - Third, Fourth, and Fifth DTC character decoding table 24
Table 13 - Mode 4 25
Table 14 - Mode 9 26
Table 15 - PID request Frame 26
Table 16 - PID Reply Frame 27
Table 17 - Examples of standard protocols and their assigned port numbers. 31
Table 18 - Mobile market share within last 4 years 40
Table 19 - Pin describtion of OBD-II to DB9 Cable 48
Table 20 - PID supported 54

Nomenclature

Table 1 - List of abbreviations and acronyms

ECU	Electronic Control Unit
CAN	Controller Area Network
OBD	On Board Diagnostic
HVAC	Heat Ventilation And Air Conditioning
HMI	Human Machine Interface
LIN	Local Interconnect Network
SAE	Society Of Automotive Engineers
CARB	California Air Resources Board
PID	Parameter Identification Numbers
VIN	Vehicle Identification Number
OEM	Original Equipment Manufacturer
IoT	Internet Of Things
ESP	Electronic Stability Program
LTE	Long Term Evolution
M2M	Machine To Machine
GPS	Global Positioning System
OSI	Open System Interconnection
HLP	Higher Layer Protocol
MAC	Medium Access Control
LLC	Logical Link Control
DTC	Diagnostic Trouble Codes
EWS	Embedded Web Server
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
DHCP	Dynamic Host Configuration Protocol

MAC	Media Access Control

Acknowledgements

"Acknowledgement and celebration are essential to
fueling passion, making people feel valid and valuable,
and giving the team a real sense of progress that makes it all worthwhile"
Attributed to ***Dwight Frindt***

Foremost, thanks and praises to ALLAH, for His floods of blessings throughout our graduation thesis to complete it successfully.

This thesis marks the culmination of an intense though incredibly gratifying journey at Cairo university faculty of engineering that started a lot of years ago. We would like to express our gratitude and thanks to our academic supervisor Dr.Hassan Mostafa for his useful comments, remarks through the learning process of this graduation thesis. Furthermore, we thank Valeo Egypt for their support to do this work. We thank Eng.Eslam Helmy for introducing us to the topic, and his support throughout our work to complete this thesis successfully.

Finally, our thanks go to all the people who have supported us to complete the thesis directly or indirectly.

We hope you find the book useful!

Abstract

We live in a connected world and people will increasingly expect to have access to applications and services on their cell phones. Obviously, there are more than 1 billion vehicle on the roads today and we rely on them in various situations in our daily lives, so the need of a connection between the cell phones and the vehicles boosted us to implement our project which called App Your Car.

App Your Car is Android mobile platform based on IoT technology. It also provides an easy way where the vehicle’s owner can monitor and diagnose it from any place. A notification will be sent to the vehicle’s owner if any error occurs, as well as the vehicle’s owner can monitor a lot of parameters in the vehicle e.g. speed, RPM, the coolant temperature of the engine, the throttle percentage of the gas pedal.

Moreover we are willing to use an important feature of the OBD-II port in the vehicle and connect through it to the CAN-BUS which is the backbone of the vehicle, and through this bus we can receive signals and data from the vehicle to monitor some readings inside the vehicle through an Android Application.

"This page intentionally left blank"

CHAPTER 1

Introduction

"As you start to walk out on the way, the way appears."
Attributed to ***Rumit***

▪ 1.1 Evolution in Vehicle

The automotive industry nowadays is the sixth largest economy in the world. Millions of vehicles are produced every year. In the last decade there was an exponential increase in embedded systems in vehicles and, more precisely in embedded software. At 1985, the cost of electric and embedded system in vehicle represented only 20% of the vehicle price, but nowadays the electric and embedded system in vehicle represent more than 60% of the vehicle price. Embedded systems in vehicles will make the vehicle perform better, more comfort and safe. Today the vehicle contains tens of electronic control units (ECUs). ECU is hardware and software consists of electronic circuits and it is controlled by microcontroller. ECU is known as the vehicle's computer. Examples of ECUs in vehicle are Engine control unit, Door control unit, Brake control unit, Airbags, Wipers …etc.

The pressure of customers for more performance, safety, and comfort will push the automotive industry to a rapid evolution in technological progress. The quality of the components of electronic technology today has become robust and reliable.

Examples of Automotive embedded systems that are now essential in every vehicle are ABS, electronic engine control, electronic stability program (ESP). These technology made the customer buy safe and efficient vehicle, also with better performance. In the last decade, several automotive embedded systems network were developed e.g. Controller area network (CAN), local interconnect networks (LIN), FlexRay, TTP/C. These automotive embedded systems networks are designed to decrease the number of wires in the vehicle as number of ECUS in vehicle has been increased dramatically. After the evolution in automotive embedded systems, autonomous cars is going to be everywhere very soon and this technology will open various of fields

1. Automated Road Vehicles
2. Automated Road Network
3. Automated Road Management

As we have seen, control technologies based on sophisticated sensors and control units running advanced algorithms are now quickly arriving in standard road vehicles

Today these technology is based on the comfort and safety of the vehicle owner. In short thanks to automotive embedded system as very soon transportation will rely less on human driver.

▪ 1.2 Evolution in Mobile

Cell phone industry today is vividly growing very fast. In the last 5 years there were huge evolutionary growth in cell phones subscribers. In 2000 cell phones subscribers was around1 billion, and it is expected to be 5.5

billion in 2018. The cell phone today is essential in everyday life of billions of people around the world, it is a small computer in their pockets .

In short thanks to technology it made our life better and faster, today people depend on technology too much, they cannot imagine their life without it. The pressure of customers push the cell phone industry for more performance and innovative devices.

Long term evolution (LTE) is a standard of wireless communication of high speed data for mobile phones and data terminals, and it is market as 4G. This generation open a new trend called internet of things (IoT), where physical things embedded with electronics can exchange data with each other. IoT will make people's life easier. LTE and IoT are expected to serve the devices, and systems that goes beyond machine to machine (M2M), and smart city where vehicles, buildings and everything embedded with electronics can communicate with each other without the intervention of human being.

▪ 1.3 Thesis Organization and Overview of Methods and Contributions

Now we will provide an overview of the contributions of each chapter, including methodologies and results, as well as an overview of the chapter structure. The introductory paragraphs of each chapter provide more detailed outlines.

The overarching theme of the thesis is the proposal of method to monitor and control the car main functions using an android application through the worldwide web (Internet).

Throughout this thesis, we provide our demonstrations of how to interface the TCP/IP protocol with automotive industry through an android app and explaining our modular and flexible blocks of building our model.

• 1.3.1 Chapter 2: Motivation

We will begin by explaining the main motivation schemes that our thesis is published to serve it. We will talk about a main trend in the world which is called Internet of Things (IoT). We will describe it and its influence on all the electronic industries especially on automotive industry nowadays. Also we will talk about the use of the Ethernet in automotive industry and find also how the Ethernet will make a development revolution in automotive industry. As there is a huge number of smartphones which are connected to the internet nowadays, so such a smartphone also can play an effective role in the automotive Ethernet technology.

• 1.3.2 Chapter 3: Overview & Background

In this chapter we will go through a brief introduction about reviewing all the technologies we used and learnt through this year to help us in demonstrating our project and finalizing the thesis in this way. Our project is based on more than one technology, we divided it into blocks, and each block based on a separate technology we learnt through the past year. We will begin from the interior of the vehicle's Electronic Control Unit (ECU) and how it is programmed to do a certain task inside the vehicle and how it is connected to the Head Unit (HU) of the vehicle. After that we will talk about the Controller Area Network (CAN) bus and how it is used to connect all the ECUs in the car together. Moreover we need to access to the car using a very important interface inside the car which is called On Board Diagnostic (OBD) port. We will talk about its standards and how we can access it through an android application. Finally, the protocol used to connect the android application to the car. And here we will use the TCP\IP protocol.

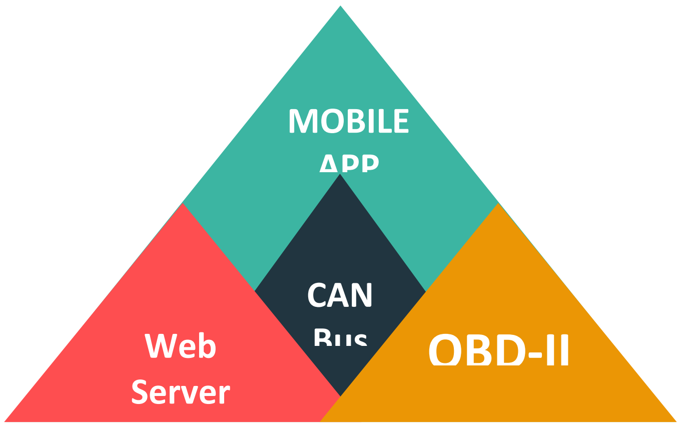


Figure 1 - The building blocks of our project

- **1.3.3 Chapter 4: Project Description & Related Work**

This chapter is divided into two main sections: related work & our project. First, we will go back for one or two years to find a previous work or a project that related to what we want to do. By performing some analysis to this old work, we can make a good use of it and find how can we add another features or improve the previous ones. Second, we will go in a nutshell through our project definition, description and what it presents to nowadays technologies. Then the components used in the project will be mentioned by their description and features. This project is a combination of building blocks that tested individually. So we will present these blocks by tasks. There are four tasks Ethernet web server, Reading real time values from different ECUs in the vehicle, Reading the DTC of the vehicle and mobile application end. We will implement every task in detail and integrate them together to achieve the final project requirements.

- **1.3.4 Chapter 5: Technical Approach**

In this chapter we will explain in details the analysis of each task of the main tasks mentioned briefly in chapter 4. First we will talk about the hardware used in every task, then the chosen library used to be compatible with the hardware kit. Then we will illustrate the important functions and classes used from to serve the task requirements.

- **1.3.5 Chapter 6: Implementation and Results**

This chapter will present the overall project implementation. By integrating all tasks mentioned in the last chapter, we can concatenate the whole program code which will be loaded to the program memory on the microcontroller. Also we will talk about how is the overall code arranged and written to achieve the project specifications with acceptable efficiency, and what are the problems that faced us and how we solved it.

- **1.3.6 Chapter 7: Conclusions and Future Work**

In this chapter we are going to show how App your Car could be upgraded. Our Project is just the cornerstone of automotive mobile technology. Also we will suggest some features that could be added to the project to improve the overall performance. We will go in a nutshell through our project to conclude and encapsulate what we have done on this project.

CHAPTER 2

Motivation

"Some people want it to happen,
Some wish it would happen, others make it happen"
Attributed to ***Michael Jordan***

▪ 2.1 Internet of Things

Internet of things is where things with embedded electronics can communicate with each other, without the requiring human to human or human to human interaction. The internet of things has the potential to make a huge change in the world, like internet did, may be even more. It is expected that mobile to machine sessions number will be 35 times higher than the number of mobile sessions.

As mention before the IoT is now leading the future technology. IoT idea is based on IPs, even Vehicle will have IPs so they can communicate with each other. IoT will open three design dimension

- Awareness as object will understand the human activity occurring in physical world.
- Interaction as object interact with user in terms of input, output, and control.
- Representation as object will be programmed for specific application.

But whatever the motivation for the automotive companies, the addition of IoT to the car will also be a boon for consumers. [1]

▪ 2.2 Automotive Ethernet

Introducing a new technology for networking in the automotive industry will be very expensive and difficult. As the huge number of vehicles which exist in our world, this means a large number of modifications if we want to accompany a new networking technology. We will have to make some extensive testing and validation, which are expensive and time-consuming. [1]

But what we will explain now, we believe Ethernet offers the new network advantage that outweigh any drawbacks associated with its adoption, we are on the cusp of a new era.

Key Information: *Ethernet is the most popular wired networking technology in the world, and was the basis for creating Wi-Fi, the most popular wireless networking technology in the world. Bringing Ethernet into vehicles paves the way for Wi-Fi, which will in turn enable vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) networking connectivity. Among the many potential benefits these can enable, V2V technologies have the potential to prevent over 3/4 of roadway accidents.*

Automotive Ethernet (AE) provides what is needed for today's applications based on IoT. The Ethernet revolution, starting now in the automotive industry, will be even larger that the Controller Area Network (CAN) revolution in 1990s.

2.3 Worlds will Collide

In 2013, BMW's X5 hit the road with the latest automotive networking technology which is the Automotive Ethernet. The obvious question that might come to mind is why after 40 years Ethernet now is used in automotive networking? Finally, it is the time now to automotive industry and Ethernet to collide together to serve the world's progress in technology. [1]

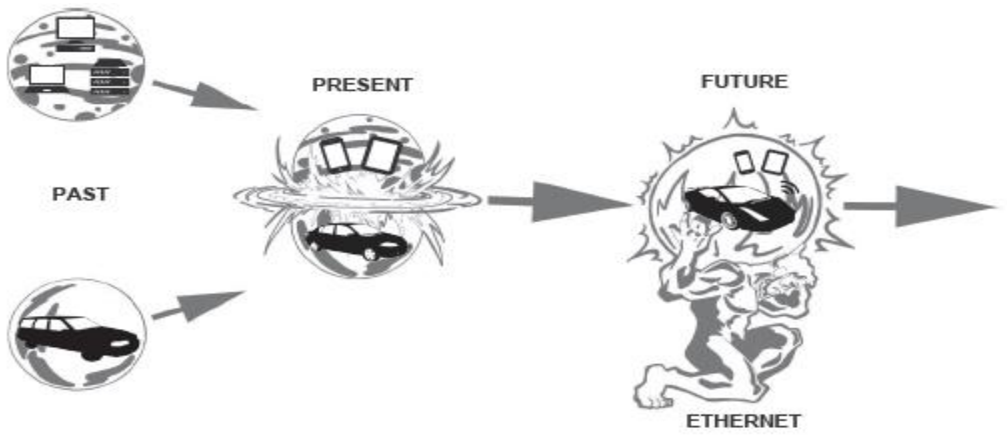


Figure 2 - in the past, automotive and ethernet technologies are separate, but now the are convergend togethe

The explosion in popularity of mobile devices also will collide with the Automotive Ethernet technology. The smart phones connected to the internet now is considered as a handheld devices and it is integrated with all our daily live tasks [1]

For these and other reasons, we are now seeing convergence of the world of automotive and Ethernet networking and smart phone applications. Using the concept of IoT which promises to improve the driving experience and save lives. However, to fully unlock this technology, a wide range of barriers need to be addressed, including security, safety, regulation, lack of cross-industry standards. [1]

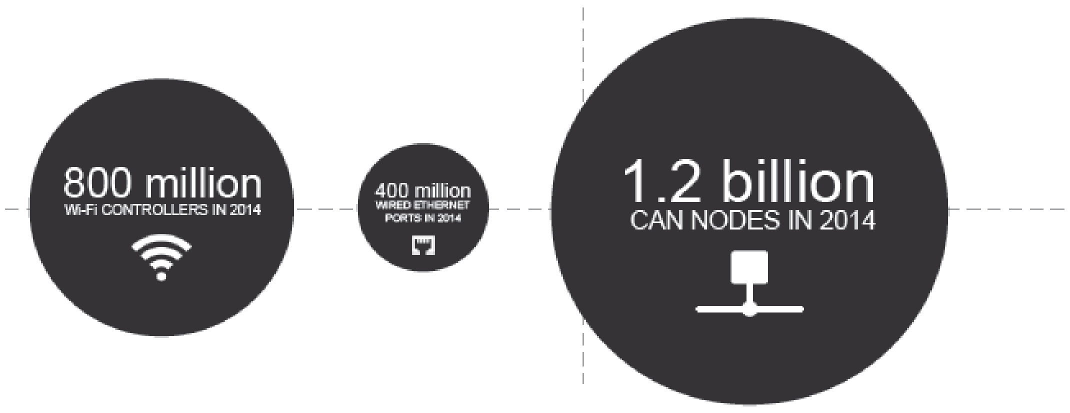


Figure 3 - As of 2014, each year approximately three times more CAN nodes ship worldwide than wired Ethernet ports, an amount roughly equal to the combined total of wired Ethernet and wireless Wi-Fi controllers.

CHAPTER 3

Overview & Background

▪ 3.1 Introduction

In this chapter we will take a high level look at automotive industry and Ethernet protocol, and discuss some of the requirements and standards behind their design and implementation. If you are not new to this field you may already be knowledgeable about much of what we have written here, but for those who are new to any of these fields they will find it an essential introduction. It would be possible to write a whole book on each of the following topics, but that would be unfeasible, instead, what we have done is writing the main information which is sufficient to provide a general overview of the most important blocks that we will deal with it in our book, and in the end of the book you will find appropriate references for further reading.

▪ 3.2 Automotive Main Building Units

• 3.2.1 Electronic Control Unit (ECU)

3.2.1.1 Definition and Importance

In automotive industry, a rapid increase in the number of vehicle components nowadays are controlled by open-loop and closed-loop systems, which consist of sensors, actuators and electronic control units (ECU) to improve safety, comfort, reduce emission and fuel consumption.

An electronic control unit (ECU) is an embedded electronic device that is known as car computer or the brain of the car because it’s the most important part in motor vehicle, basically there is no one single computer but multiple ones which are networked with shared bus system called Controller Area Network (CAN) bus .

It reads signals which are coming from large number of sensors placed at different parts of the vehicle and based on this information, it makes different actions to control various units.

3.2.1.2 Types of ECU

There are a large number of different types of ECUs. The high percentage of electrical to mechanical in automotive industry is the reason for the massive number of ECU’s in today vehicle. Some modern motor vehicles have up to 80 ECUs.

Depending on what it is used for, ECUs are named and differentiated as

- Engine control unit (ECU)
- Speed control unit (SCU)
- Brake Control Module (ABS)
- Door control unit (DCU)
- Airbag control
- Human machine interface (HMI)

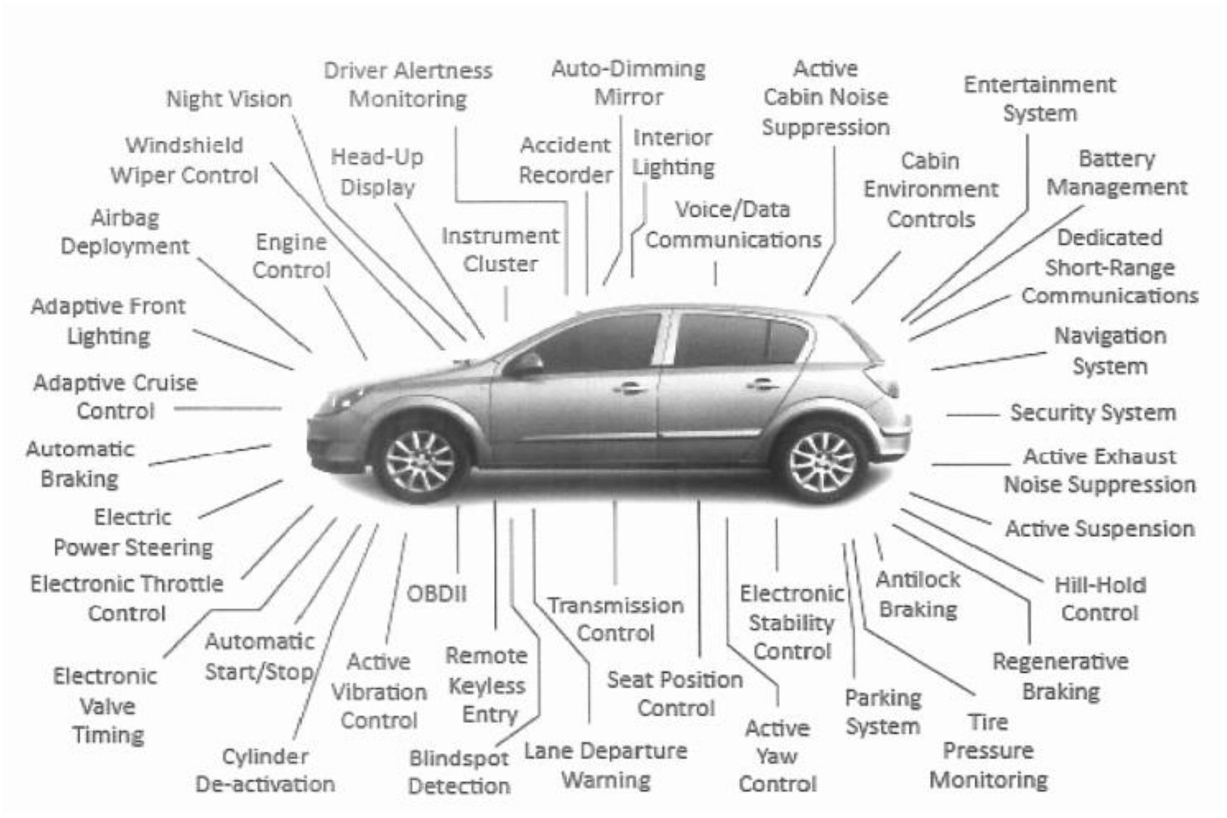


Figure 4 - A photo represents the huge number of ECUs inside a vehicle nowadays

Handling the increasing complexity, large types and number of ECUs in a vehicle has become a challenge key for Original Equipment Manufacturers (OEMs). Figure 4 shows the large number of different ECUs.
For example, if the driver wants to increase the speed of the vehicle by put more pressure on the fuel throttle, that pressure is sensed, converted into electrical signal and forwarded to the electronic control unit to process it and order the fuel injection pump to pump more fuel to the vehicle engine according to this pressure. As a result of this high rotation speed of gears will occur and consequently wheels run faster. This cycle is shown in figure 4.

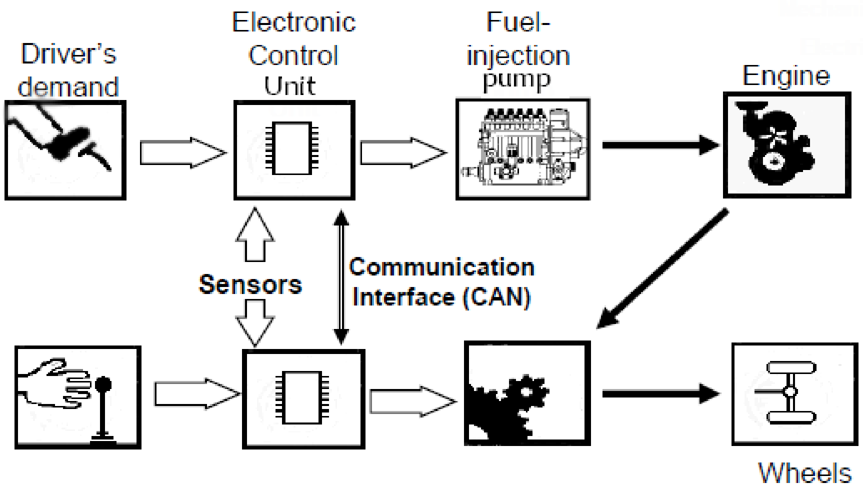


Figure 5 - Cycle of CAN bus

3.2.1.3 ECU Architecture

The architecture of the ECU includes hardware and software development to accomplish the required function from it depending on where the ECU is placed.

Automotive ECU's are being developed following the V-model which represents a software development process and demonstrates the relationships between each phase of the development life cycle and its associated phase of testing.

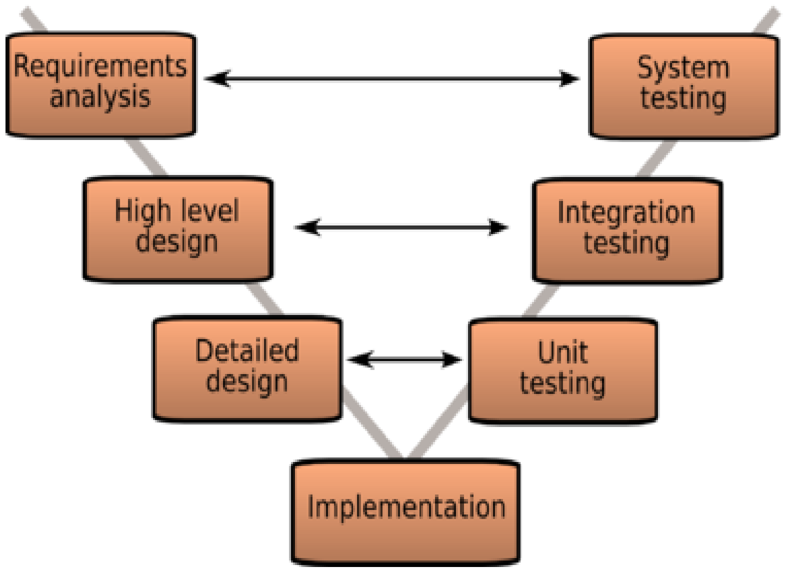


Figure 6 - V-Model

First the requirements of the overall system must be considered and the high level design will be performed. Second the system is divided into sub-systems or sub-processes, the process becomes very low-level design to the point of uploading code onto target processor chips (Implementation).

Testing every sub-system individually to ensure that it is working well under all conditions and performs its desired function. After that, sub-systems are combined and tested together until such time that the entire system can enter final testing to achieve requirements and that is typically what is used in ECU design cycle.

3.2.1.4 Hardware Architecture of a Typical ECU

The hardware is made up of different electronic components on a printed circuit board (PCB). The most important components are

- Microcontroller chip which fetches, decodes and executes the instructions as coded and generates output signals by processing the information using specified control algorithms.
- Memories which are volatile (RAM) for data and non-volatile (ROM) for program memory.
- Input/output interface in order to interact with external devices like sensors and actuators.

3.2.1.5 Software Architecture of a Typical ECU

The software is lower-level language that runs on the microcontroller and saved on the non-volatile program memory. The lower level language contains commands or functions in the language set that closely to processor instructions.

In the automotive industry, managing complexity of Electronics architectures is one of the big challenges. So as to manage rising system complexity and increasing number of electronic devices while keeping the cost practical.

Standardizing the main software and the interfaces to bus systems and applications in a future proof way is a significant issue to leading automotive suppliers and manufactures. Therefore they lunched the Automotive Open System Architecture (AUTOSAR) development partnership in 2003.

The software standard AUTOSAR standardizes the ECU software architecture, which raises the reuse of ECU software components between vehicle suppliers and manufacturers plus their use within various vehicle platforms.

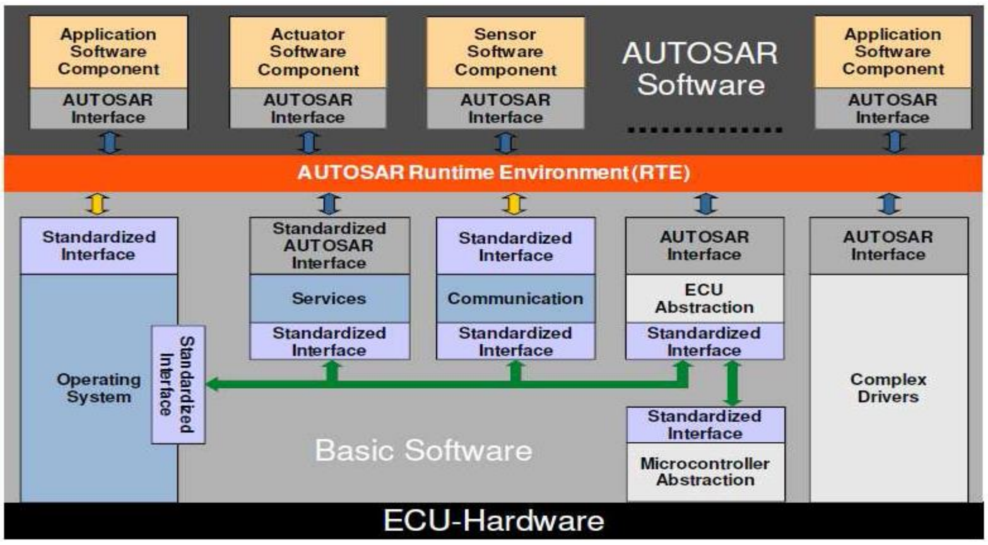


Figure 7 - AUTOSAR software architecture

The layered architecture of the AUTOSAR provides all the mechanisms needed for software and hardware independence. It distinguishes between three primary software layers which run on a Microcontroller: Application layer, Runtime Basic Software (BSW) and Environment (RTE). The Basic Software layers are branched to functional groups, for example System, Memory and Communication Services.

Software Components which are defined by AUTOSAR as the application software that runs on the ECU. Software components make up the technology that hosts the application software. A software component is an abstraction that allows the function of parts of application software to be encapsulated this encapsulation decreases the element dependency on the surroundings in which it executes. This implies that a software component have to run on any ECU compatible with AUTOSAR. Although the layered software architecture of the AUTOSAR provides the independence of the software components. It aims to increase the reuse of software components, specifically between different vehicle platforms, and between suppliers and OEMs.

So far what is the ECU, what is it used for, how it is manufactured and programmed are explained, then thinking about how would this large number of ECUs work at the same time on one shared bus between them is an essential question? There must be some protocols and priorities to manage which will get the bus and if there are more than one ECU try to get the bus which will be served first and why?

• 3.2.2 Controller Area Network (CAN)

3.2.2.1 Introduction to CAN

For more than 30 years, CAN has been the dominant network in the automotive industry. There are very good reasons why CAN has been so successful to be one of the main building block in the automotive industry. CAN, which stands for Controller Area Network, is a low cost and simple implemented multicast-based serial communication protocol .it was developed in the mid1980s by Bosch to provide a cost-effective communications bus for automotive applications. [2]

With the growing demand for greater safety and comfort for reduced fuel consumption and the comfort of the passengers, the car industry has developed a plenty of electronic systems (e.g. ABS, EMS, air-bags, central door locking …). The complexity of these systems and the need to send and receive data among them requires more signal busses to transfer the signals through them. That is the main reason CAN was invented for. All controllers, sensors, and actuators communicate with each other, in real-time over a broadcast digital bus. [3]

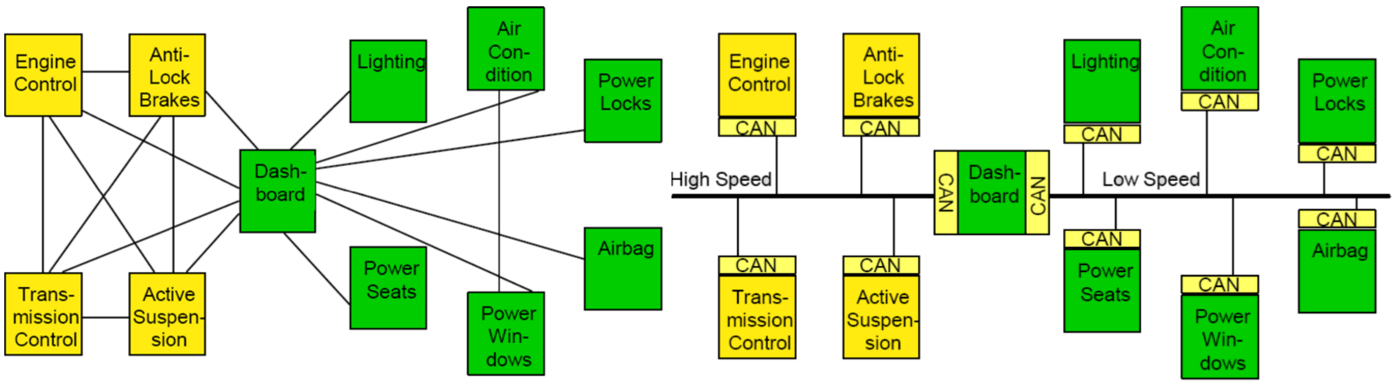


Figure 8 - The diagram on the left represents how the car ECUs were connected before using CAN bus, and the right connection is after using the CAN bus

CAN is not limited for automotive industry but it is also used in factory and plant controls, in robotics, and also in some aviation systems due to its attractive solution for embedded control systems because of advantages that we will talk about later in this chapter. [2]

In this chapter, we will discuss and cover all aspects of the design and analysis of a CAN communication system.

3.2.2.2 CAN Bus Topology

CAN bus is a standardized multi-master serial bus used in the automotive industry for connecting Electronic Control Units (ECUs) also known as nodes. All these nodes are connected to each other through a two nominal twisted pair wire bus. [4]

The interface between a CAN chip and a differential bus typically consists of a transceiver amplifier. The transceiver converts the electrical representation of a bit from the one in use by the controller to the one defined for the bus. Then, it must provide adequate output current to protect the chip from overloading and drive the electrical state of the bus. [2]

And from the receiver point of view, the CAN transceiver provides the recessive signal level and protects the controller chip input comparator against any overloaded voltages on the bus. Moreover, the receiver detects errors in the bus such as line breakage, short circuits, etc. it also can be used in the galvanic isolation between a CAN node and the bus line. [2]

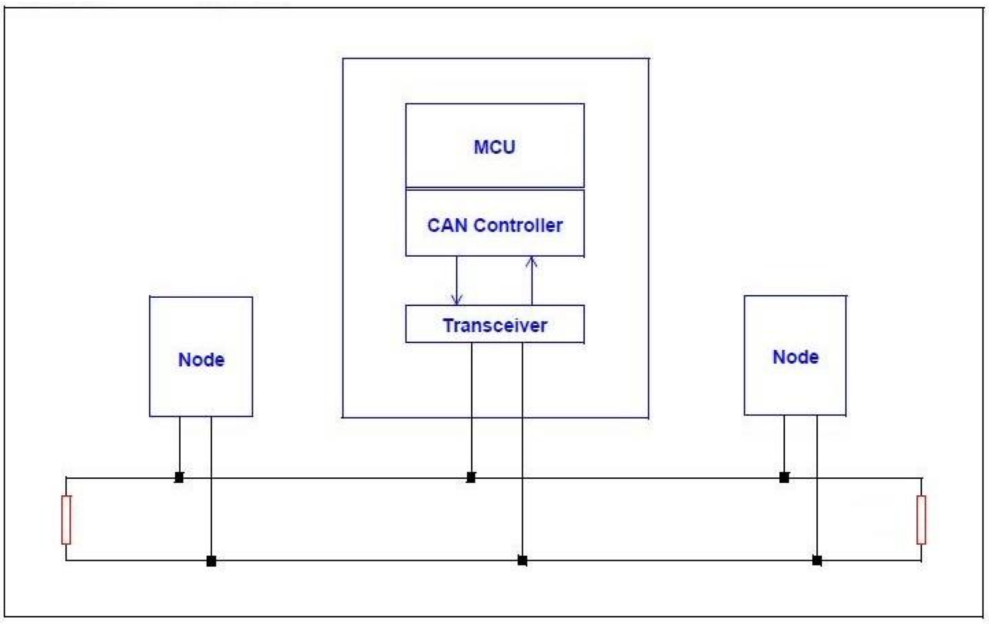


Figure 9 - CAN Bus

3.2.2.3 CAN Standard Specifications

Almost all of the network protocols are organized using the seven layer Open System Interconnection (OSI) model. CAN bust protocol defines the Data Link Layer and part of the Physical Layer in the OSI model. The remaining physical layer (and all of the higher layers) are not defined by the CAN specification. These remaining layers can be defined by the system designer, or they can be implemented using non-proprietary Higher Layer Protocols (HLPs) and physical layers. [5]

The main function of the data link layer is putting the signals received from the physical layer into a message to provide a procedure for data transmission control such as transmission error control. Some functions of the data link layer is executed in the hardware by the CAN controller are assembling messages into a frame, returning ACK, and detecting errors. [6]

The Logical Link Control (LLC) is responsible for managing the overload control and notification, message filtering and recovery management functions. The Medium Access Control (MAC) performs the data encapsulation/de-capsulation, error detection and control, bit stuffing/de-stuffing and the serialization and de-serialization functions. [5]

Inside the physical layer, the protocol is responsible for defining the manner in which signals are transmitted and the bit encoding, and synchronization procedures. However, this does not mean that the signal levels, communication speeds, sampling point values, driver and bus electrical characteristics, and connector forms are defined specifically by the CAN protocol. [6]

The International Standards Organization (ISO) has defined a standard of the CAN as well as the physical layer. The standard, ISO-11898, was originally created for high-speed in vehicle communications using CAN. ISO-11898 specifies the physical layer to ensure compatibility between CAN transceivers. A CAN controller typically implements the entire CAN specification in hardware, as shown in figure 9.

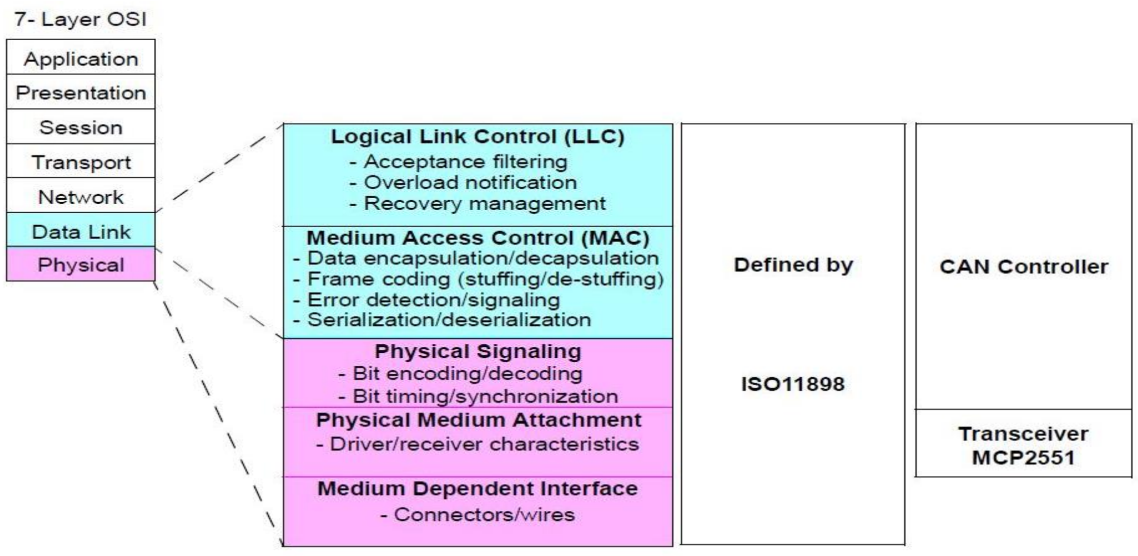

7- Layer OSI
Application
Presentation
Session
Transport
Network
Data Link
Physical
Logical Link Control (LLC)
- Acceptance filtering
- Overload notification
- Recovery management
Medium Access Control (MAC)
- Data encapsulation/decapsulation
- Frame coding (stuffing/de-stuffing)
- Error detection/signaling
- Serialization/deserialization
Physical Signaling
- Bit encoding/decoding
- Bit timing/synchronization
Physical Medium Attachment
- Driver/receiver characteristics
Medium Dependent Interface
- Connectors/wires
Defined by
ISO11898
CAN Controller
Transceiver
MCP2551

Figure 10 - CAN bus and OSI model

CAN specifies two logical states: recessive state and dominant state. According to ISO-11898, a differential voltage is used to represent recessive and dominant states (or bits), as shown in Figure 2.6. The differential voltage in the recessive state (logic 1) on CAN H and CAN L is less than the minimum threshold (<0.5V receiver input or <1.5V transmitter output). In the dominant state (logic 0), the differential voltage on CAN H and CAN L is greater than the minimum threshold. If at least a node outputs a dominant bit, the status of the bus changes to dominant regardless of other recessive bit outputs. This is the foundation of the nondestructive bitwise arbitration of CAN.

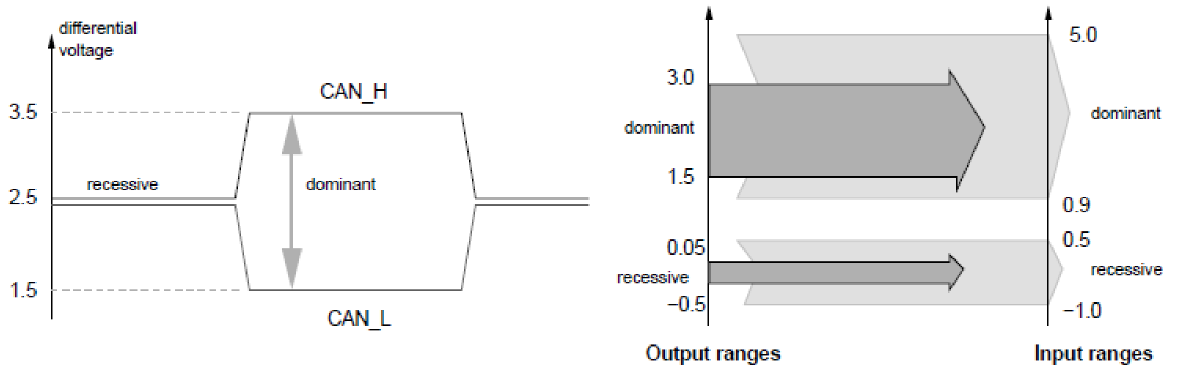


Figure 11 - Encoding of dominant and recessive bits

ISO included CAN in its specifications as ISO 11898, with three parts: ISO 11898- ISO 11898-2 (high speed CAN) and ISO 11898-3 (low-speed or fault-tolerant CAN). ISO standards include the PMA and MDA parts of the physical layer. The most common type of physical signaling is the one defined by the CAN ISO 11898-2 standard, a two-wire balanced signaling scheme. ISO 11898-3 defines another two-wire balanced signaling scheme for lower bus speeds. It is fault tolerant, so the signaling can continue even if one bus wire is cut or shorted.

3.2.2.4 Message Frames

Communication is performed using the following five types of frames:
• Data frame
• Remote frame
• Error frame
• Overload frame
• Interframe space
Of these frames, the data and the remote frames need to be set by the user. The other frames are set by the hardware part of CAN. The data and the remote frames come in two frame formats: standard and extended. The standard format has an 11-bit ID as shown in figure 11, and the extended format has a 29-bit ID as shown in figure 12. Roles of each frame are listed in table 2.

Frame	Roles of frame	User settings
Data frame	This frame is used by the transmit unit to send a message to the receive unit	Necessary
Remote frame	This frame is used by the receive unit to request transmission of a message that has the same ID from the transmit unit	Necessary
Error frame	When an error is detected, this frame is used to notify other units of the detected error.	Unnecessary
Overload frame	This frame is used by the receive unit to notify that it has not been prepared to receive frames yet.	Unnecessary
Interframe space	This frame is used to separate a data or remote frame from preceding frame.	Unnnecessary

Table 2 - Roles of each frame in Message frames

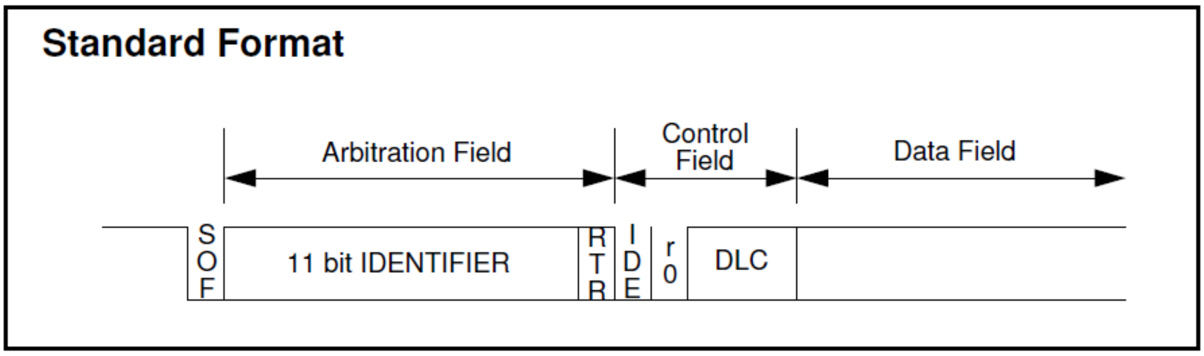


Figure 12 - CAN standard Arbitration Field

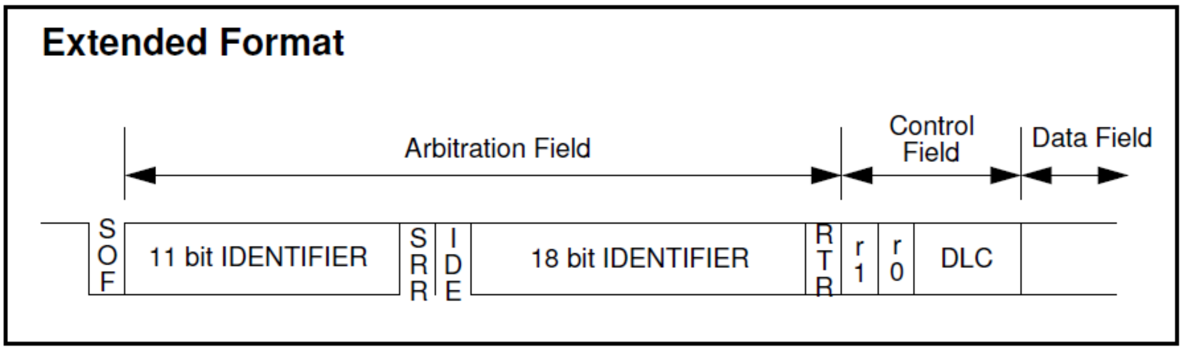


Figure 13 - CAN Ectended Arbitration Field

3.2.2.5 Data Frame

It is used by the transmit unit to send a message to the receive unit, and is the most fundamental frame handled by the user. The data frame consists of seven fields. Figure 13 shows the structure of the data frame.

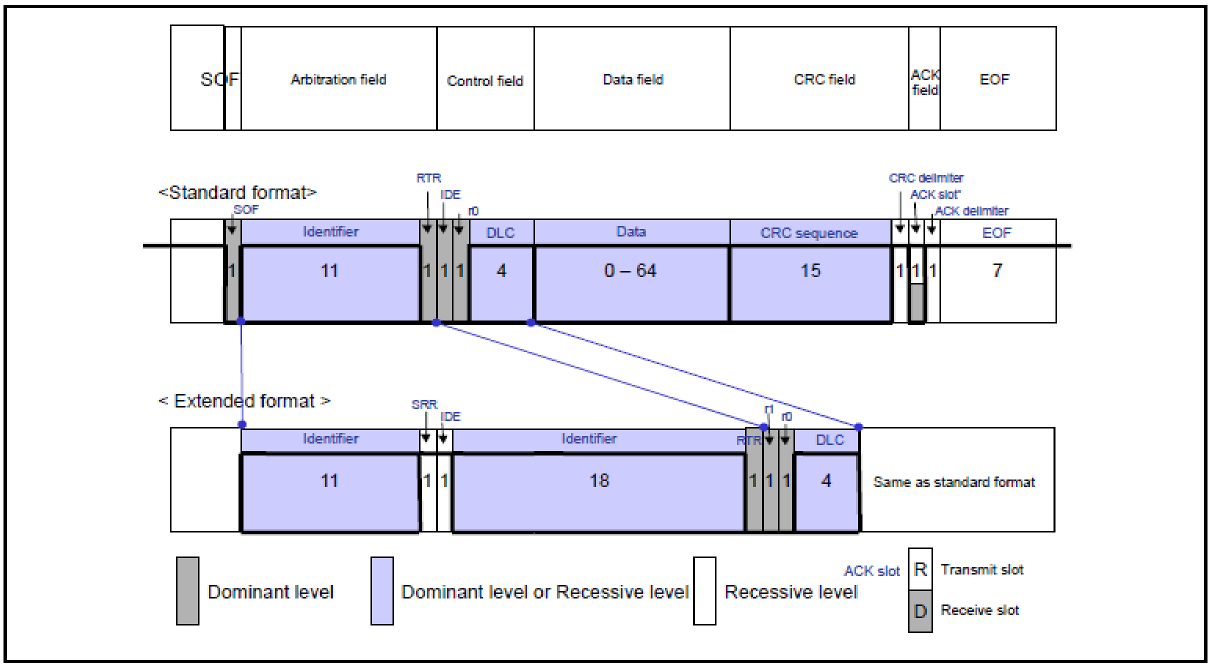


Figure14 - Structure of the Data Frame

Frame	Description
Start of Frame	This field indicates the beginning of a data frame.
Arbitration Field	The field indicates the priority of a frame.
Control Field	This field indicates reserved bits and the number of data bytes.
Data Field	This is the content of data. Data in the range 0 to 8 bytes can be transmitted.
CRC Field	This field is used to check the frame for a transmission error.
ACK Field	This field indicates a signal for confirmation that the frame has been received normally.
End of Frame	This field indicates the end of the frame.

Table 3 - Discreption of the frames in Data Frame

Let's move on inside each of this frames to know what it is consists of

1. **Start of frame:** This field indicates the beginning of a frame. It consists of one dominant bit.

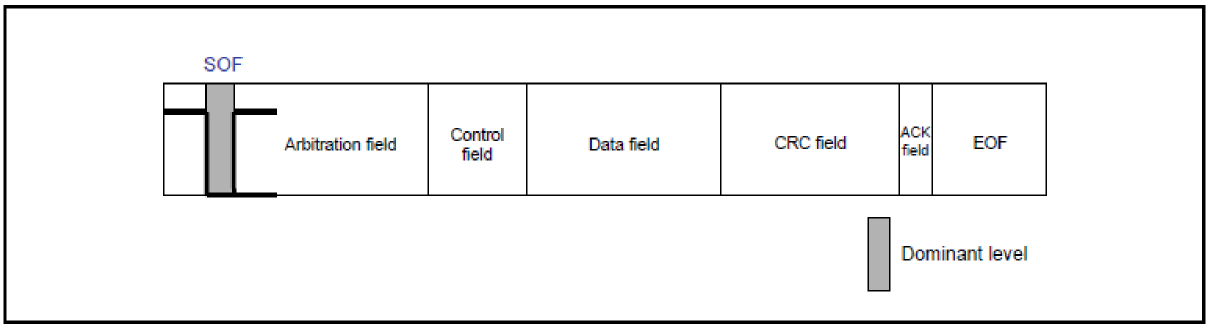


Figure 15 - Data Frame (Start of Frame)

2. **Arbitration Field:** The field indicates the priority of a frame.

The standard format ID consists of 11 base ID bits (ID28–ID18). These ID bits are transmitted sequentially beginning with ID28. The 7 high-order bits cannot all be recessive. (IDs set to 1111111XXXX are prohibited.) Up to 2,032 discrete IDs can be set. The extended format ID consists of 11 base ID bits (ID28–ID18) and 18 extended ID bits (ID17–ID0). The base ID is the same as in the standard format. The 7 high-order bits cannot all be recessive. (IDs set to 1111111XXXX are prohibited.) Up to 2,032 × 218 discrete IDs can be set. In no case can multiple units on the bus transmit data frames with the same ID at the same time.

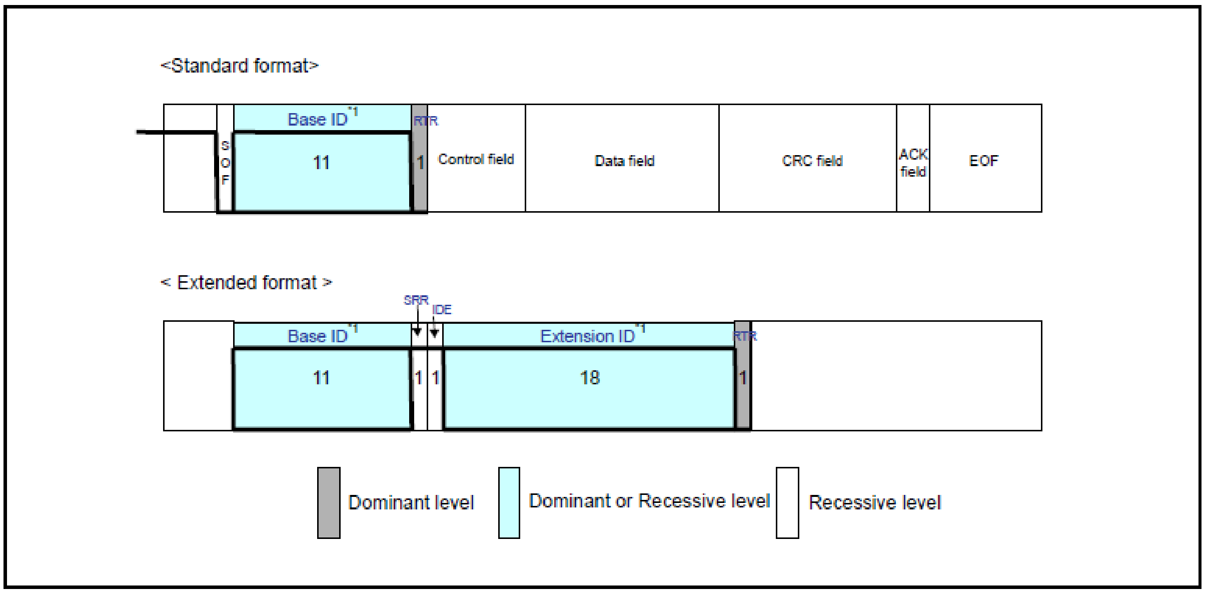


Figure 16 - Data Frame (Arbitration Field)

3. **Control field:** This field indicates the number of data bytes in a message to be transmitted. The structure of this field differs between the standard and extended formats.

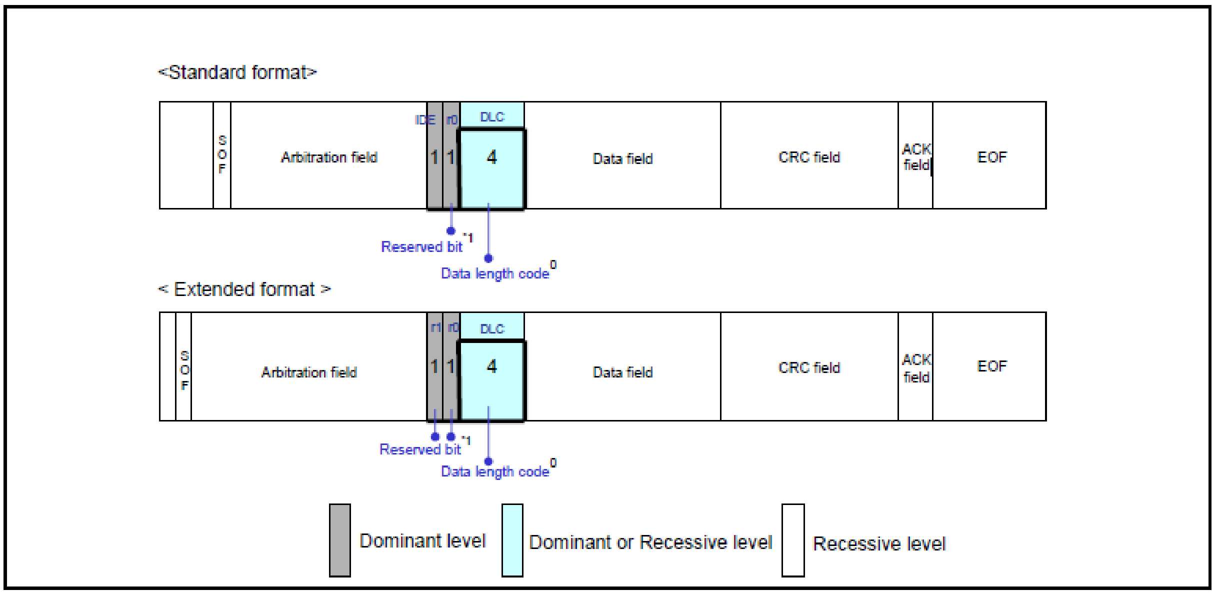


Figure 17 - Data Frame (Control Field)

1: Reserved bits (r0, r1)
Each reserved bit will be transmitted with a domain level. On the other side the receiving side, they will be received in any combination of domains.

2: Data length code (DLC)
In table 4 the data length code and the numbers of data bytes will be shown.

Number of data bytes	**Data length code**			
	DLC3	*DLC2*	*DLC1*	*DLC0*
0	D	D	D	D
1	D	D	D	R
2	D	D	R	D
3	D	D	R	R
4	D	R	D	D
5	D	R	D	R
6	D	R	R	D
7	D	R	R	R
8	R	D OR R	D OR R	D OR R

Table 4 - Data Length Code DLC of Control Frame

4. **Data field**: indicates the content of data. Zero to 8 bytes of data set in the control field can be transmitted. The data is output beginning with the MSB side.

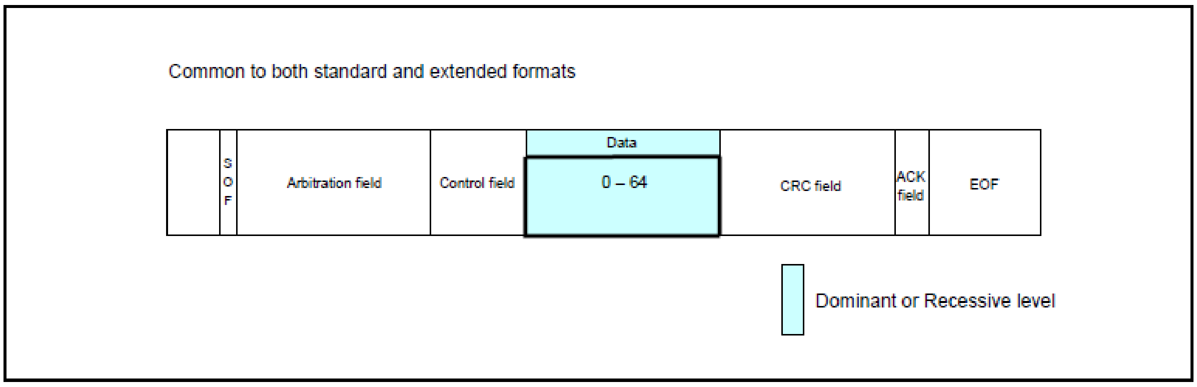

Common to both standard and extended formats
S O F
Arbitration field
Control field
Data
0 – 64
CRC field
ACK field
EOF
Dominant or Recessive level

Figure 18 - Data Frame (Data Field)

5. **CRC field** (common to both standard and extended formats): CRC field used to check the transmission error frame. It consists of a 15-bit CRC sequence and a 1-bit CRC delimiter (separating bit).

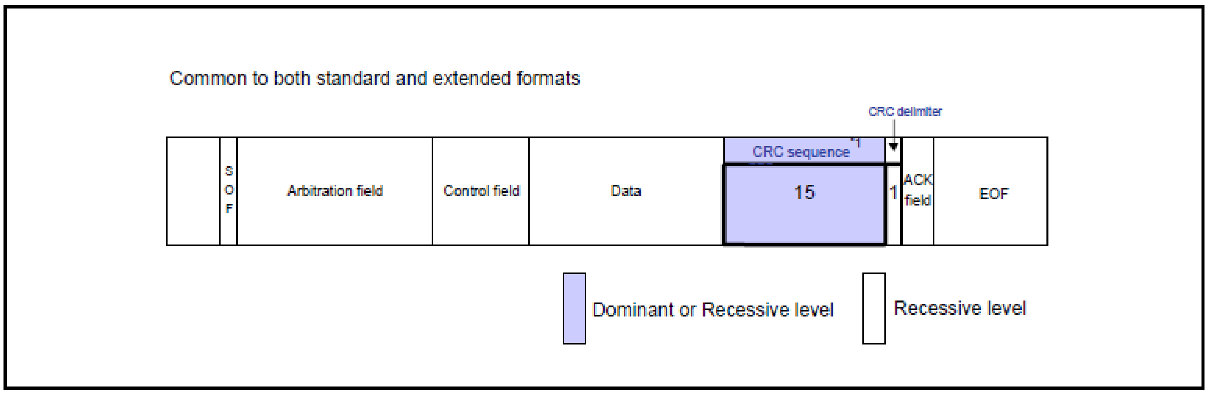


Figure 19 - Data Frame (CRC Field)

CRC sequence

The CRC sequence is consisting of a value which is generated by the polynomial P(X). The CRC calculation range contains the start of frame, arbitration field, control field, and data field. On the receive side, CRC value is calculated in the same range of fields, after that calculated CRC and the received CRC are compared then if they match that means no error occurred, otherwise an error is occurred

The CRC calculation P(X) = $X^{15} + X^{14} + X^{10} + X^{8} + X^{7} + X^{4} + X^{3} + 1$

6. **ACK field:** ACK field indicates a signal to confirm if the frame has been received or not. ACF field consists of 2 bits, the first is for ACK slot and the other for ACK delimiter.

1) ACK field of the transmit unit

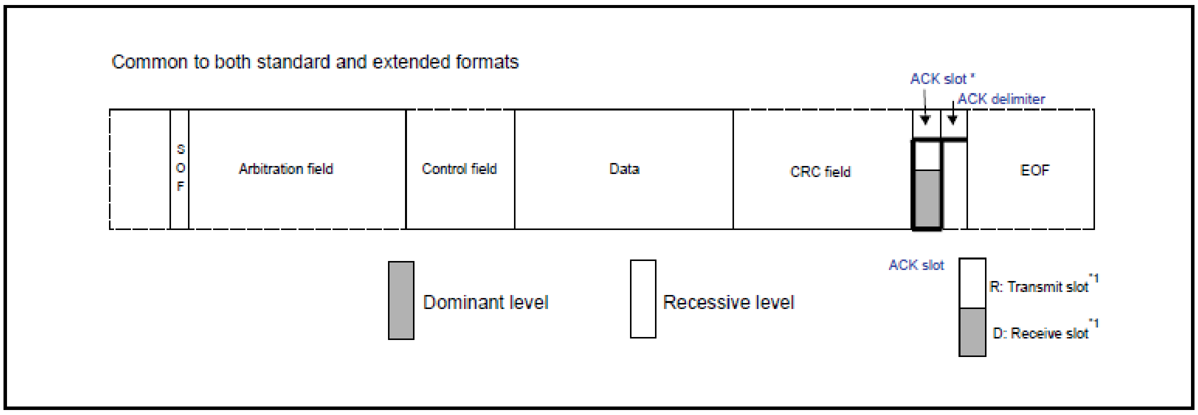


Figure 20 - Data Frame (ACK Field)

The transmit unit sends the ACK slot and ACK in recessive bits.

2) ACK field of the receive unit

When the receive unit received a correct message, it sends a dominant bit in the ACK slot of the received frame to notify the transmit unit that it has successively finished the receiving normally. This is referred to as “sending ACK” or “returning ACK.”

7. **End of Frame:** End of frame indicates the end of the frame, and it consists of 7 recessive bits only.

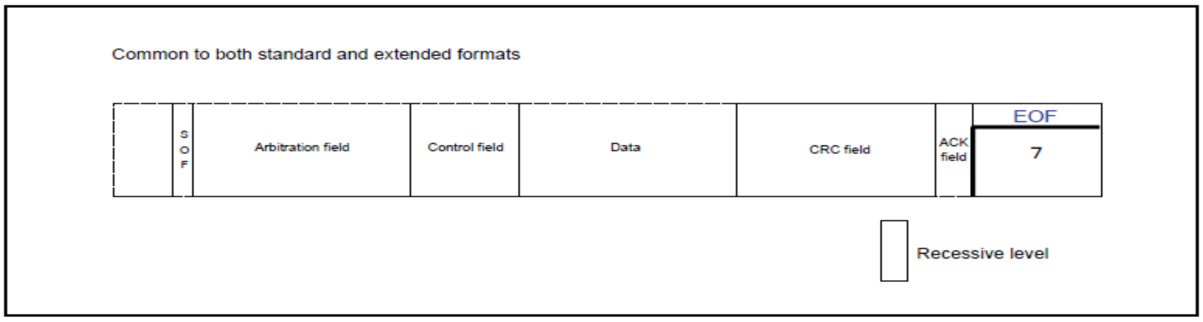


Figure21 - Data Frame (End of Frame)

3.2.2.6 Remote Frame

Remote frame used by receiver, this field used to request transmitting a message from the transmit unit. It consists of six fields. It is very similar to data frame except for it doesn't have data field. Figure 21 shows the structure of the remote frame.

Frame	Description
Start of Frame	This field indicates the beginning of a data frame.
Arbitration Field	The field indicates the priority of a frame.
Control Field	This field indicates reserved bits and the number of data bytes.
Data Field	This is the content of data. Data in the range 0 to 8 bytes can be transmitted.
CRC Field	This field is used to check the frame for a transmission error.
ACK Field	This field indicates a signal for confirmation that the frame has been received normally.
End of Frame	This field indicates the end of the frame.

Table 5 - The frames of the Remote Frame

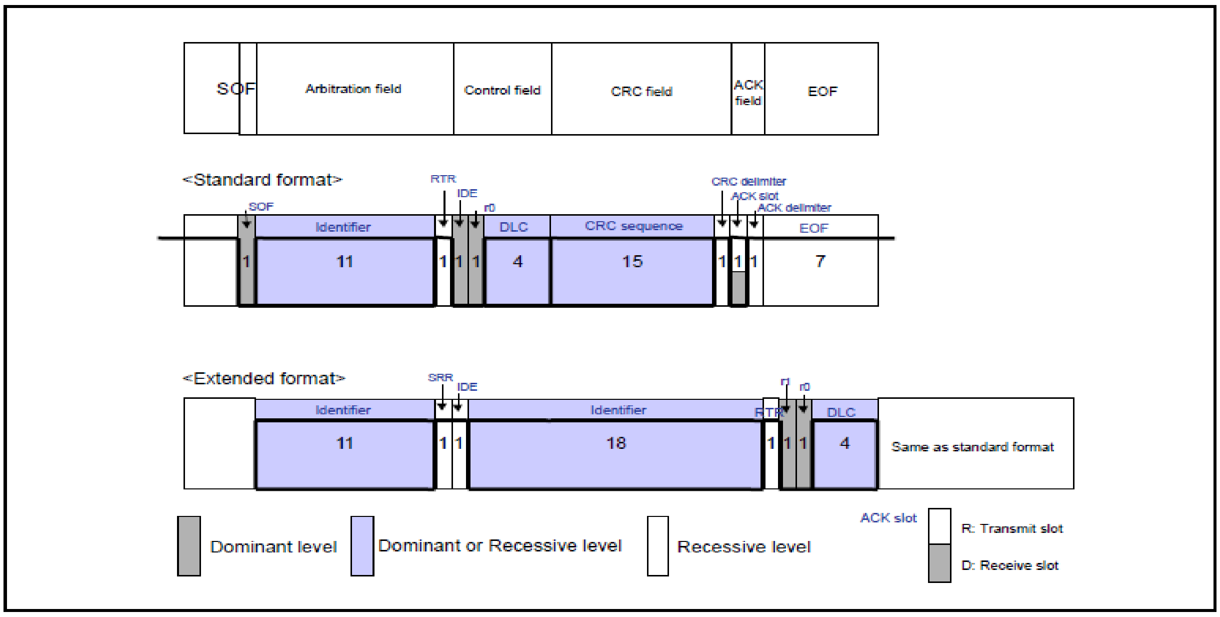


Figure 22 - Structure of the Remote Frame

3.2.2.7 Error Frame

This frame is to give a notification that an error has been occurred during transmission. It frame consists of an error delimiter and an error flag. They are transmitted by the hardware part of CAN. Figure 22 shows the structure of the error frame.

1. Error flag

There are two types of error flags: passive-error flag and active-error flag.
- Passive-error flag: 6 recessive bits
- Active-error flag: 6 dominant bits

2. Error delimiter

It consists of 8 recessive bits.

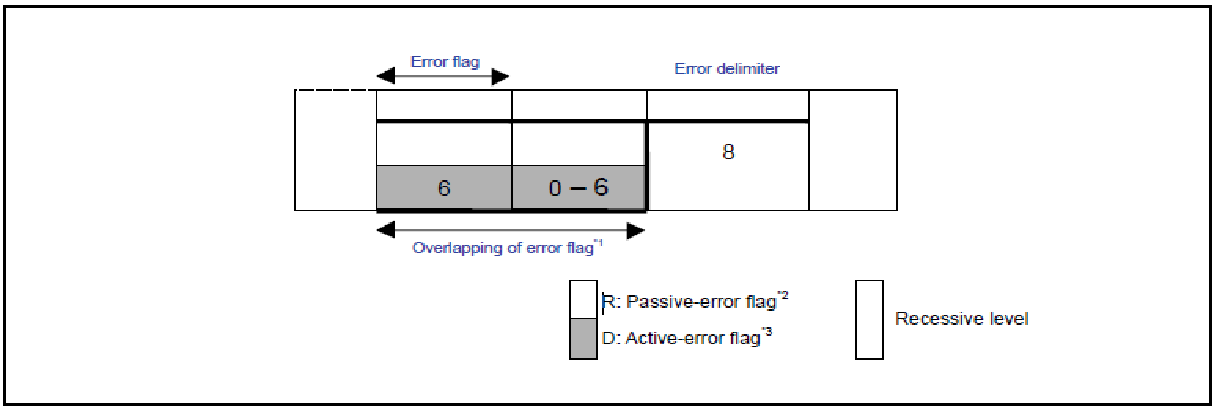


Figure 23 - Error Frame

a) Error flag overlapping
It depends on the timing when error is detected by unit connected to the bus. It may overlap, it is up to 12 bits in total length.
b) Passive-error flag
This error flag is output by a unit in an error-passive state when it detected an error.
c) Active-error flag
This error flag is output by a unit in an error-active state that it detected an error.

3.2.2.8 Overload Frame

The overload frame is used by the receive unit to give a notification that is not ready to receive frames. It consists of an overload delimiter and an overload flag. Figure 23 shows the structure of the overload frame.

a) Overload flag
Overload flag consists of 6 dominant bits. The structure of it is similar to active error flag
b) Overload delimiter
Overload delimiter consists of 8 recessive bits. The structure of it is similar to error delimiter.

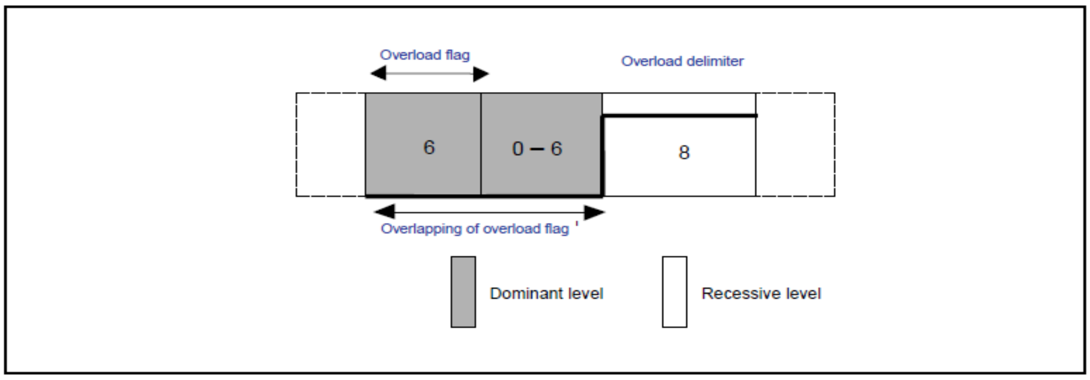


Figure 24 - Structure of the Overload Frame

3.2.2.9 Interframe Space

Interframe space is used to separate the remote frames and the data. The data and the remote frames may have been transmitted earlier to them, and are separated from the preceding frame by an inserted interframe space. However, no interframe spaces are inserted preceding overload and error frames. Figure 24 shows the structure of the interframe space.

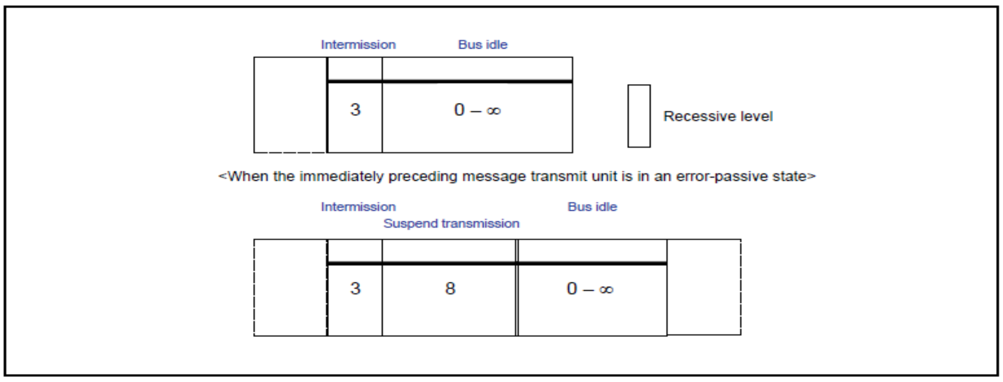


Figure 25 - Structure of the Interframe Space

a) Intermission

Intermission consists of 3 recessive bits. When a dominant level is detected during an intermission, then overload frame will be transmitted. However, if the third bit of an intermission is of a dominant level, the intermission is recognized as the SOF.

b) Bus idle

Bus idle consists of a recessive level. It has unlimited length. In this state, the bus is thought to be free, and any transmit unit can start sending a message.

c) Suspend transmission (transmission pause period)

Suspend transmission consists of 8 recessive bits. This field is included in only an interframe space if the immediately preceding message transmit unit was in an error-passive state.

• 3.2.3 On Board Diagnostics

3.2.3.1 What is OBD?

On-board diagnostics (OBD) is an automotive term related to the self-diagnosis and self-reports of the vehicle. OBD gives the owner or the technician access to some of the subsystems of the vehicle. It is located in most of cars on the road today. Through years, OBD systems become more sophisticated. It is standardized by specifying the type of diagnostic connector and its pinout, the electrical signaling protocols available, and the messaging format. [7]

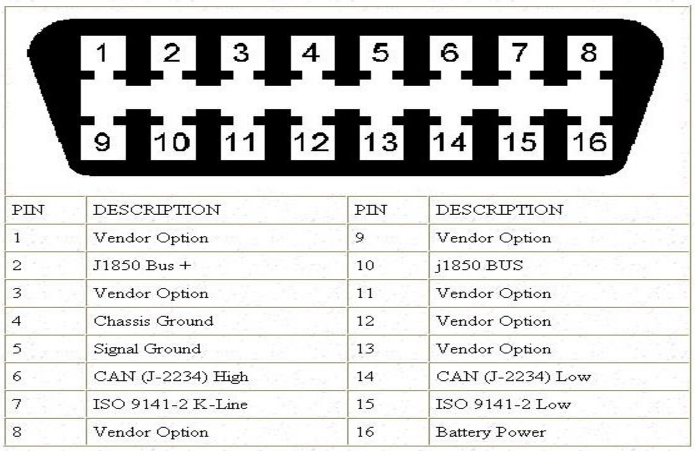
Figure 26 - OBD connector

3.2.3.2 History of OBD-II?

During the early 80's, manufacturers started to immerse electronic components to control the vehicle functions and diagnose its problems. In 1991 The California Air Resources Board required that all vehicles sold in California from this date and forwards must have OBD. Later this interface was named as OBD-I after 3 years from 1991 when OBD-II specifications was issued by CARB. In 1996, The OBD-II specification is made obligatory for all vehicles manufactured in the USA or sold there. [8]

3.2.3.3 OBD-II Connector

OBD connector is a 16-pin cable connector which is connected to the OBD-II interface in the car. Each pin has its own connection which is illustrated in figure 25.



PIN	DESCRIPTION	PIN	DESCRIPTION
1	Vendor Option	9	Vendor Option
2	J1850 Bus +	10	j1850 BUS
3	Vendor Option	11	Vendor Option
4	Chassis Ground	12	Vendor Option
5	Signal Ground	13	Vendor Option
6	CAN (J-2234) High	14	CAN (J-2234) Low
7	ISO 9141-2 K-Line	15	ISO 9141-2 Low
8	Vendor Option	16	Battery Power

Figure 27 - OBD connector and Pinout

OBD-II gives access to the information inside the engine control unit (ECU) and also gives valuable information about troubleshooting problems inside the engine and the vehicle itself. The Society of Automotive Engineers (SAE) standardize the OBD-II and defined methods to request diagnostic data and the list of the parameters that may be available to be read from the ECU in real time monitoring.

Those parameters are addressed by Parameter Identification Numbers (PIDs) which is a list of some numbers in Hexadecimal format as shown in table 6. Manufactures are not required to implement all the PIDs which are stated in table 7 the can include proprietary PIDs which are not listed.

3.2.3.4 Standard PIDs

A								B							
A7	A6	A5	A4	A3	A2	A1	A0	B7	B6	B5	B4	B3	B2	B1	B0

C								D							
C7	C6	C5	C4	C3	C2	C1	C0	D7	D6	D5	D4	D3	D2	D1	D0

- **PIDs Modes:**

Mode (Hex)	Description
01	Show current data
02	Show freeze frame data
03	Show stored Diagnostic Trouble Codes
04	Clear Diagnostic Trouble Codes and stored values
05	Test results, oxygen sensor monitoring (non CAN only)
06	Test results, other component/system monitoring (Test results, oxygen sensor monitoring for CAN only)
07	Show pending Diagnostic Trouble Codes (detected during current or last driving cycle)
08	Control operation of on-board component/system
09	Request vehicle information

Table 6 - All available PID modes (the two orange shaded modes are the most important modes)

- **Mode 1:**

It is used to capture the current data of the vehicle through can bus and OBD-II interface. It has a lot of PID request addresses which is written in Hexadecimal, but we will only care about some of them which are written in the following table 7 and the rest of them will be found in APPENDIX I [9]

PID (Hex)	Data Bytes returned	Description	Min Value	Max Value	Units
00	4	PIDs supported [01 - 20]			
05	1	Engine coolant temperature	-40	215	ºC
10	2	MAF air flow rate : The airflow rate as measured by the mass air flow sensor.	0	655.35	gm/sec
0C	2	Engine RPM	0	16,383.75	rpm
0D	1	Vehicle Speed	0	255	Km/h
14	2	Oxygen sensor voltage	0	1.275	volts
11	1	Throttle position: Usually above 0% at idle and less than 100% at full throttle	0	100	%
51	1	Fuel Type			

Table 7 - Mode 1 PIDs

Fuel type: Mode 1 PID 51 returns back a value from the following list which indicates the fuel type. The response is a single byte, and the value is given by the following table: [9]

Value	Description	Value	Description	Value	Description
0	Not Available	8	Electric	16	Bifuel running electric and combustion engine
1	Gasoline	9	Bifuel running Gasoline	17	Hybrid gasoline
2	Methanol	10	Bifuel running Methanol	18	Hybrid Ethanol
3	Ethanol	11	Bifuel running Ethanol	19	Hybrid Diesel
4	Diesel	12	Bifuel running LPG	20	Hybrid Electric
5	LPG	13	Bifuel running CNG	21	Hybrid running electric and combustion engine
6	CNG	14	Bifuel running Propane	22	Hybrid Regenerative
7	Propane	15	Bifuel running Electricity	23	Bifuel running diesel

Table 8 - Fuel Type response

- **Mode 2:**

Mode 02 accepts the same PIDs as mode 01, with the same meaning, but information given is from when the freeze frame was created. [9]

PID (Hex)	Data Bytes returned	Description
02	2	DTC that caused freeze frame to be stored.

Table 9 - Mode 2 message formation

- **Mode 3:** (Request Trouble Codes)

Lists the emission-related "confirmed" diagnostic trouble codes stored. It displays exact numeric, 4 digit codes identifying the faults. A request for this mode returns a list of the Diagnosis Trouble Codes (DTCs) that have been set. The list is defined using the ISO 15765-2 protocol.

These fault codes are standard for all makes of vehicle and are divided into 4 categories:

1. P : Powertrain
2. C : Chassis
3. B : Body
4. U : Network

Mode 3 actually does not have any PIDs. All that is sent is the three header bytes, the mode byte, and the cyclic redundancy check (CRC). And the vehicle will respond with header bytes, a mode byte (which is 43 – mode 3 + 40h offset), six data bytes, and finally the CRC or checksum byte for the message.

The six data bytes following the mode byte define up to 3 trouble codes. Each trouble code is represented with two bytes. If there are less than 3 trouble codes, you will still get all six bytes, but some of them will be a pair of zeros, which means no trouble code in that position. This is done to keep the message length constant. If the vehicle has not stored any trouble codes, it probably won't even respond to the mode 3 request. If it has more than three codes stored, it will send multiple message frames, and each of them can hold up to 3 codes.

Below are definitions so you can decode these data bytes.

A7-A6	First DTC character
00	P
01	C
10	B
11	U

Table 10 - First DTC character decoding table

A5-A4	Second DTC character
00	0
01	1
10	2
11	3

Table 11 - Second DTC character decoding table

A3-A0	3rd, 4th, & 5th DTC character
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Table 12 - Third, Fourth, and Fifth DTC character decoding table

Let's do an example to use the above tables. Assume we get the following data bytes from a vehicle:
Data A = 20 hex = 00100000 binary
Data B = 90 hex = 10010000 binary
Data C through Data F are 00 hex

By decoding the bits using the above tables

A7 A6 = 00 which represents the letter P
A5 A4 = 01 which represents the numeral 1
A3 A2 A1 A0 = 0100 which represents the numeral 4
B7 B6 B5 B4 = 1001 which represents the numeral 9
B3 B2 B1 B0 = 0101 which represents the numeral 5

So, the trouble code in this example happens to be a P1495, which is a TCSPL solenoid circuit malfunction. [11]

- **Mode 4:**

Is used to clear emission-related diagnostic information. This includes clearing the stored pending/confirmed DTCs and Freeze Frame data. [9]

PID (Hex)	Data Bytes returned	Description
N\A	2	Clear trouble codes / Malfunction indicator lamp (MIL) / Check engine light

Table 13 - Mode 4

- **Mode 5:**

Displays the oxygen sensor monitor screen and the test results gathered about the oxygen sensor [9]

- **Mode 6:**

Is a request for on-board monitoring test results for continuously and non-continuously monitored system. There are typically a minimum value, a maximum value, and a current value for each non-continuous monitor. [9]

- **Mode 7:**

Is a request for emission-related diagnostic trouble codes detected during current or last completed driving cycle. It enables the external test equipment to obtain "pending" diagnostic trouble codes detected during current or last completed driving cycle for emission-related components/systems. This is used by service technicians after a vehicle repair, and after clearing diagnostic information to see test results after a single driving cycle to determine if the repair has fixed the problem. [9]

- **Mode 8:**

Could enable the off-board test device to control the operation of an on-board system, test, or component.

- **Mode 9:**

PID (Hex)	Data Bytes returned	Description
00	4	Mode 9 supported PIDs (01 to 20)
02	17-20	Vehicle Identification Number (VIN)
04	16	Calibration ID : ID for the software installed on the ECU
06	4	Calibration Verification Numbers (CVN)
08	4	In-use performance tracking for spark ignition vehicles

Table 14 - Mode 9

In-use performance tracking for spark ignition vehicles: It supplies the needed information about track-in-use performance for catalyst banks, oxygen sensor banks, evaporative leak detection systems, EGR systems and secondary air system.

3.2.3.5 Non Standard PIDs

All OEMs use standard and non-standard PIDSs. This database contains manufacturer specific codes. Where manufacturers feel that a code is not available within the generic list, they can add their own codes. The definitions for these are set by the manufacturer. [9]

3.2.3.6 PID Request & Response

The PID request and response occurs through the CAN bus. Standard requests and responses use certain addresses. A request is initiated using CAN ID $7DF, which is a broadcast address, and accepts responses from any ID in the range $7E8 to $7EF. ECUs that can respond to OBD queries listen both to the functional broadcast ID of $7DF and one assigned ID in the range $7E0 to $7E7.

This technique allows up to 8 ECUs, each is responding alone to OBD requests. The reader can use the ID in the ECU response frame to communicate with a specific ECU. In particular, multi-frame communication requires a response to the specific ECU ID rather than to ID $7DF.

CAN bus may also be used for communication beyond the standard OBD messages. Physical addressing uses particular CAN IDs for specific modules (e.g., 720 for the instrument cluster in Fords) with proprietary frame payloads.

PID Type	Byte							
	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
SAE Standard 7DF	Number of additional data bytes: 2	Mode 01: show current data Mode 02: freeze frame Etc.	PID code(e.g.: 05 = Engine coolant temperature)	not used (may be 55h)				

Table 15 - PID request Frame

The vehicle responds to the PID request with message IDs varies from one module to another. Typically the engine or main ECU responds at ID 7E8h. Other modules, like the hybrid controller or battery controller in a Prius, respond at 07E9h, 07EAh, 07EBh, etc. The message uses 8 data bytes regardless (CAN bus protocol form Frame format with 8 data bytes). The bytes are

PID Type	**Byte**							
	0	***1***	***2***	***3***	***4***	***5***	***6***	***7***
SAE Standard 7E8h, 7E9h, 7EAh,	Number of additional data bytes: 3 to 6	Custom mode Same as query, except that 40h is added to the mode value. So: 41h = show current data; 42h = freeze frame; etc.	PID code (e.g.: 05 = Engine coolant temperature)	value of the specified parameter, byte 0	value, byte 1 (optional)	value, byte 2 (optional)	value, byte 3 (optional)	not used (may be 00h or 55h)

Table 16 - PID Reply Frame

▪ 3.3 Embedded Web Server

• 3.3.1 Introduction

Many of embedded systems have been isolated, at most, they might have communicated with other systems within a limited range on a local network. Nowadays, this case is a trail of the past. Embedded systems increasingly use the Internet as a way to communicate with each other and with the people managing them. [13]

Figure 28 - Embedded Web applications take the Internet to places previously unheard of – even a tank.

In figure 27, it is clear that we see a military tank is this figure, a beast of a device which makes the ground tremble before it even moves. Our job is to control and position the turret with high accuracy in an enviroment which is not welcoming a human or any necessary electronics. Vibration, high tempratures, and explosives. [14]

However, if we need to solve this problem with less cost, then we have to introduce the Embedded Web Server (EWS). What if you could replace the main display in the tank with a simple PC and have the controllers of the tank hydraulics, turret, cameras, and other devices communicate over LAN? This kind of approach cuts costs and translates into many other application areas. [14]

If we take intelligent homes as another smaller example, the have to be connected to the Internet and require a microcontroller to communicate with the other network devices, as shown in figure 28. [15]

Even more importantly, embedded systems, like those in the tank and intelligent homes, are increasingly deployed in remote locations. The cost of components in comparison to the cost of delivery to remote places for breakdowns or maintenance. When you're facing these odds, web-based applications offer more versatility and less component cost, and can take advantage of LAN or Internet infrastructure. [14]

As we said before, using the Internet to control an embedded system has a direct impact on the cost of building and maintaining the system. Physical controls like buttons mean additional components and cost as well as a design that must accommodate access to the controls.

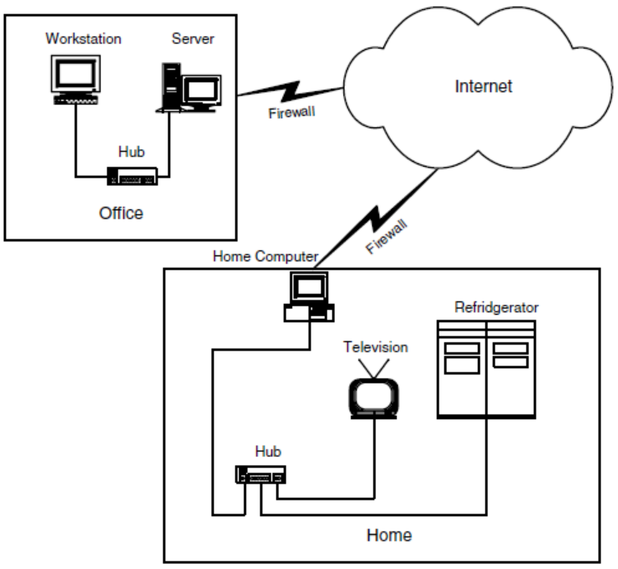


Figure 29 - Monitoring Home equipments in an inteligent home through Internet

Even more importantly, embedded systems are increasingly being deployed in remote locations. In those applications, the cost isn't dominated by components, but rather by the people that must go out to maintain and operate the system. This makes remote access critical to the cost-effectiveness of the system. The Internet is the most straightforward way of getting to the target system because worldwide infrastructure already exists. [13]

This means that an embedded system must include a built in integrated web connectivity so that it can respond to browser requests. This could be done using a compination of hardware components and a good written software.

- **3.3.2 Web Servers**

In our proposed system, the main function to get access on a sensor or an actuator that is attached to our hardware via Web browser or mobile application is the embedded web server. Where, such EWS is used to bring the desired orders and pictures over the worldwide Internet or a local network to the Web browser. In this section, the implementation of such EWS will be explained with the configuration details. We will begin by describing all the elements used in the network to implement the EWS. [16]

• 3.3.3 The Elements of a Network

All networks in the world have some things in common. Every network must have the physical components that enable the connected devices in the network to exchange data. And in every network, the computers must agree about how to share the data path that connects the computers, to help ensure that transmitted data gets to its destination. [17]
You can think of the modules used in networking as being stacked in layers, one above another. A computer's network protocol stack consists of the modules involved with networking.
Figure 29 shows an example of network stack for a computer that connects to an Ethernet network and supports common Internet protocols. [17]

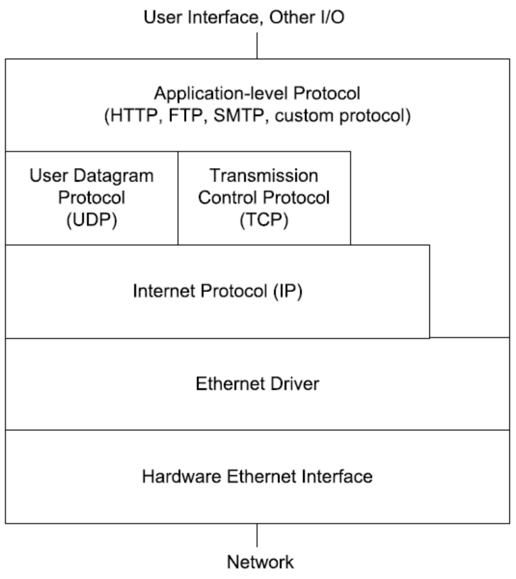

User Interface, Other I/O
Application-level Protocol
(HTTP, FTP, SMTP, custom protocol)
User Datagram Protocol (UDP)
Transmission Control Protocol (TCP)
Internet Protocol (IP)
Ethernet Driver
Hardware Ethernet Interface
Network

Figure 30 - Example of a Network layered model which supports Ethernet Communications

At the bottom of the stack is the hardware interface between two or more end user devices that need to communicate with each other. In the networks described in this book, one of the end user devices is our kit and the other end is the mobile device. On the other side of the stack – at its top-, there are the module or modules that provide data to send on the network and use the data received from the network. In the middle, there may be module or modules which are responsible for addressing, error checking and control information.

In transmitting, the message travels down through the stack from the application layer that initiates message to the network interface which puts the message on the network. In receiving, the message travels up the stack from the network interface to the application layer that uses the data in the received message. [17]

The number of layers that a message passes through in a single journey between transmitter and receiver is not constant as it can vary from time to time depending on some parameters. For some messages that travel only within a local network, the application layer can communicate directly with the Ethernet driver. Messages that travel on the Internet must use the Internet Protocol. Messages that use the Internet Protocol can also use the User Datagram Protocol (UDP) or the Transmission Control Protocol (TCP) to add error checking or flow-control capabilities. [17]

• 3.3.4 Protocols

3.3.4.1 Application-Level Protocol (Providing and Using Network Data)

The application provides data to transmit data to the network and uses data received from the network. An application often has a user interface that enables users to request data from a computer on the network or provide data to send on the network. In an embedded system, the user interface may just enable basic configuring and monitoring functions, while the system performs its network communications without user intervention.
The data which is sent or received may be a single byte; a line of text; a request for a Web page; the contents of a Web page; a file containing text, an image, binary data, or program code; or anything that the other end user requests. The data sent must follows a protocol which is a set of rules that is decoded at the received end to understand what to do with the received data.

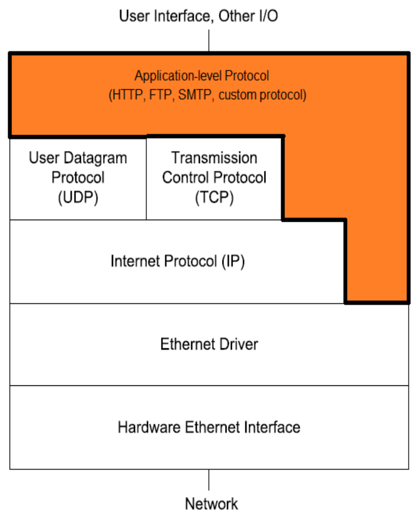

User Interface, Other I/O
Application-level Protocol
(HTTP, FTP, SMTP, custom protocol)
User Datagram Protocol (UDP)
Transmission Control Protocol (TCP)
Internet Protocol (IP)
Ethernet Driver
Hardware Ethernet Interface
Network

Figure 31 - Application layer in the Network protocol stack

There are more than one protocol used in this layer. As an example, the Hypertext Transfer Protocol (HTTP) for requesting and sending Web pages, the File Transfer Protocol (FTP) for transferring files, or the Simple Mail Transfer Protocol (SMTP) or Post Office Protocol (POP3) for e-mail messages.
In an embedded system, the application might be a module (e.g. sensor-actuator) that reads and stores sensor readings or the states of other external signals. [17]

3.3.4.2 Hypertext Transfer Protocol (HTTP)

- **Technical overview**

In our project we are concerned about the Hypertext Transfer Protocol (HTTP). Which is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.
HTTP functions as a request-response protocol in the client-server model. A web browser, for example, may be the client and an application running on a computer hosting a web site may be the server. The client submits an HTTP request message to the server. The server, which provides resources such as HTML files and other content, or performs

Figure 32 - URL beginning with the HTTP scheme and the WWW domain name label.

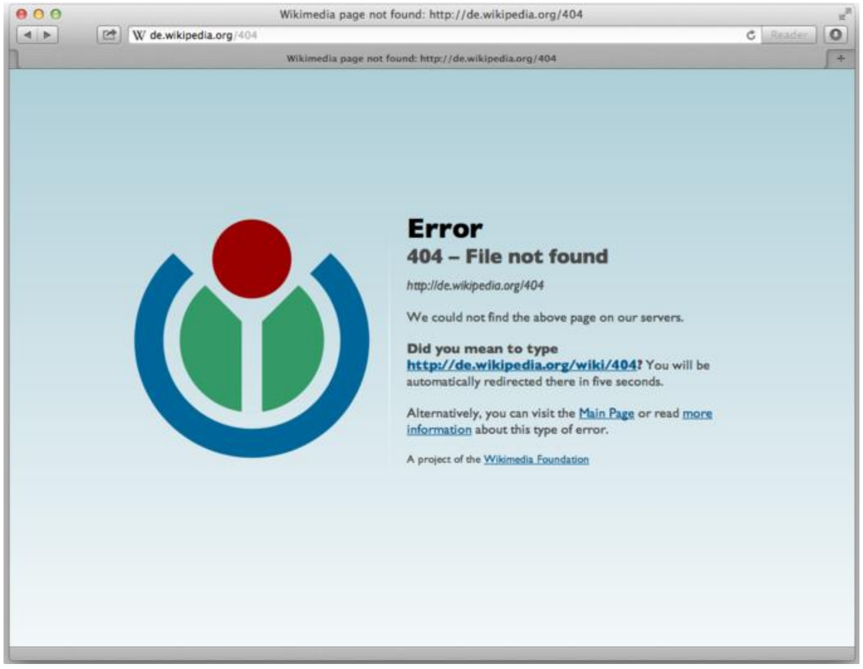
other functions on behalf of the client, returns a response message to the client. The response contains information of the completion status about the request.
HTTP is one of the application-layer protocols which is designed within the framework of the Internet Protocol Suite. It assumes a reliable transport layer protocol where (TCP) is commonly used.

- **HTTP session**

It is a sequence of network request-response transactions. First of all, An HTTP client initiates a request by establishing a (TCP) connection to a particular port on a server (a list of standard assigned port number are listed in table 17). An HTTP server listening on that port waits for a request message from a client. After receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and its own message. The body of this message is typically the requested resource, although an error message may also be returned, such as "HTTP/1.1 404 Not Found".

Protocol	Port Number
DNS	17
Telnet	23
SMTP	25
HTTP	**80**
POP3	110

Table 17 - Examples of standard protocols and their assigned port numbers.


Wikimedia page not found: http://de.wikipedia.org/404
de.wikipedia.org/404
Reader
Wikimedia page not found: http://de.wikipedia.org/404

Error
404 – File not found
http://de.wikipedia.org/404
We could not find the above page on our servers.
Did you mean to type http://de.wikipedia.org/wiki/404? You will be automatically redirected there in five seconds.
Alternatively, you can visit the Main Page or read more information about this type of error.
A project of the Wikimedia Foundation

Figure 33 - 404 error on German Wikipedia

- **HTTP Request Methods**

HTTP defines some methods to can indicate the action which will be taken. What this resource represents, whether pre-existing data or data that is generated dynamically, depends on the implementation of the server. The HTTP/1.0 specification defined the GET, POST and HEAD methods and the HTTP/1.1 specification added 5 new methods: OPTIONS, PUT, DELETE, TRACE and CONNECT. Any client can use any method and the server can be configured to support any combination of methods. If an unknown method is used it will be treated as an unsafe method. There is no limit to the number of methods that can be defined and this allows for future methods to be specified without breaking existing infrastructure.

In our project we are concerned only with GET and POST methods.

GET

Requests using GET should only retrieve data and should have no other effect. The W3C has published guidance principles on this distinction, saying, "*Web application design should be informed by the above principles, but also by the relevant limitations.*"

- **POST**

Requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The data POSTed might be, for example, an annotation for existing resources; a message for a bulletin board, newsgroup, mailing list, or comment thread; a block of data that is the result of submitting a web form to a data-handling process; or an item to add to a database.

- **Request message**

The request message consists of the following sentences:

- A request line, `GET /images/logo.png HTTP/1.1`,
  Which requests a resource called `/images/logo.png` from the server.

- Request header fields, such as `Accept-Language: en`
- An empty line.
- An optional message body.

- **Response message**

The response message consists of the following:

- A Status-Line, which include the status code and reason message. (e.g., `HTTP/1.1 200 OK`, which indicates that the client's request succeeded)
- Response header fields, such as `Content-Type: text/html`
- An empty line
- An optional message body

- **Example session**

Below is a sample conversation between an HTTP client and an HTTP server running on www.example.com, port 80.

- **Client request**

```
GET /index.html HTTP/1.1
Host: www.example.com
```

- **Server Response**

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
ETag: "3f80f-1b6-3e1cb03b"
Content-Type: text/html; charset=UTF-8
Content-Length: 138
Accept-Ranges: bytes
Connection: close
<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World, this is a very simple HTML document.
</body>
</html>
```

3.3.4.3 TCP and UDP

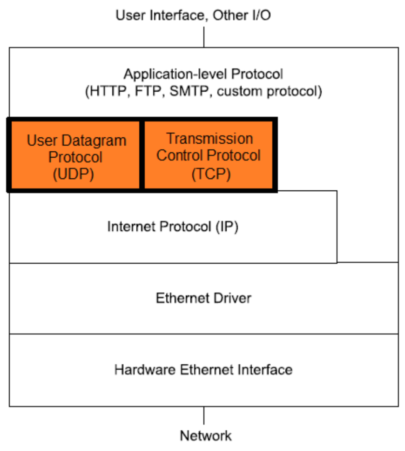

User Interface, Other I/O
Application-level Protocol
(HTTP, FTP, SMTP, custom protocol)
User Datagram Protocol (UDP)
Transmission Control Protocol (TCP)
Internet Protocol (IP)
Ethernet Driver
Hardware Ethernet Interface
Network

Figure 34 - TCP & UDP in the Network protocol stack

Any network communication must include information to help data to get to its final destination through its desired path without errors, and this is the main rule of Transmission Control Protocol (TCP) in the network stack. TCP can add information for use in error checking, flow control, and identifying an application-level process at the source and destination computers. [18]

Error checking is used in networking to detect whether the received data is corrupted or not. Flow control information help the transmitter to determine the receiver is idle or ready for receiving more data.
Many internet and local communications which requests web pages or emailing functions use TCP. Most of the Operating Systems in the market have support for TCP built in. Development kits for network-capable embedded systems often include libraries or packages with TCP support.
When sending data using TCP, the TCP layer creates a TCP segment that consists of header followed by the application data as shown in figure 30. [17]

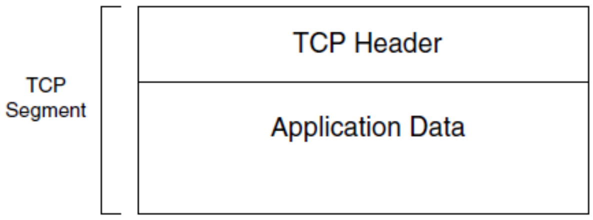
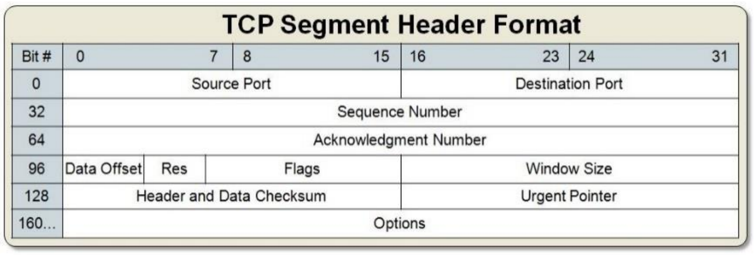


Figure 35 - TCP add a header over the data

The header is a pre-defined fields containing information used in error checking, flow control, and routing the message to the correct port at the destination.
The TCP layer doesn't affect or change the message which is carried inside TCP segment. It just places the message (encapsulates) that came from the application layer in the data portion of the TCP segment. The TCP layer then passes the segment to the IP layer for transmitting on the network.
In the other direction, the TCP layer receives a segment from the IP layer, strips the TCP header, and passes the segment to the port specified in the TCP header. [19]



TCP Segment Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset, Res	Flags	Window Size	
128	Header and Data Checksum		Urgent Pointer	
160...	Options			

Figure 36 - TCP Header Format

A simpler alternative to TCP is the User Datagram Protocol (UDP). Like a TCP segment, a UDP datagram has a header, followed by a data portion that contains the application data. UDP includes fields for specifying ports and optional error-checking, but no support for flow control. Windows and many development kits for embedded systems include support for UDP. [20]

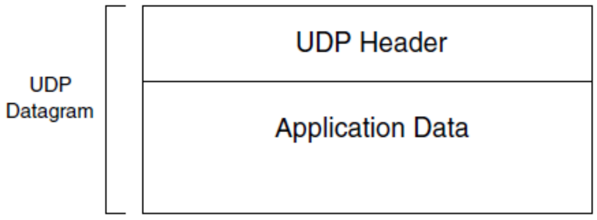
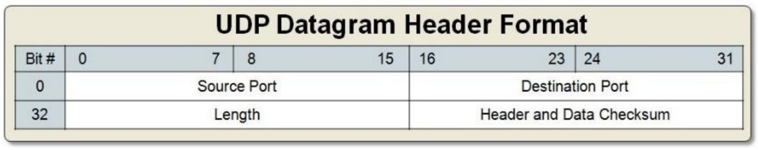


Figure 37 - UDP add a header over the data

In some networks, communications may skip the TCP/UDP layer entirely. For example, a local network of embedded systems may have no need for flow control or additional error-checking beyond what the Ethernet frame provides. In these cases, an application may communicate directly with a lower layer in the network protocol stack, such as the IP layer or Ethernet driver.



UDP Datagram Header Format

Bit #	0 7	8 15	16 23	24 31
0	Source Port		Destination Port	
32	Length		Header and Data Checksum	

Figure 38 - UDP Header Format

3.3.4.4 Internet Protocol Address (IP)

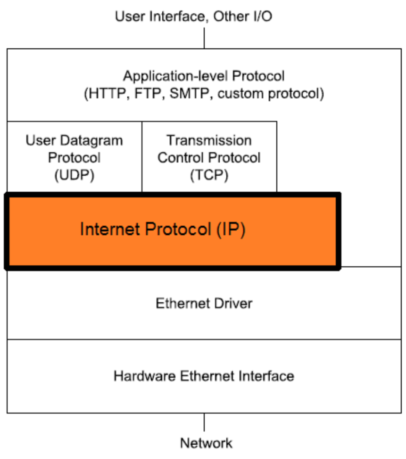

User Interface, Other I/O
Application-level Protocol (HTTP, FTP, SMTP, custom protocol)
User Datagram Protocol (UDP)
Transmission Control Protocol (TCP)
Internet Protocol (IP)
Ethernet Driver
Hardware Ethernet Interface
Network

Figure 39 - IP in the Network protocol stack

The Internet Protocol (IP) layer can help data get to its destination even if the source and destination computers are on different local networks. As the name suggests, the Internet Protocol enables computers on the Internet to communicate with each other. Because IP is closely tied to TCP and UDP, local networks that use TCP and UDP also use IP.

The term TCP/IP refers to communications that use TCP and IP. The term can also refer more broadly to the suite of protocols that includes TCP, IP, and related protocols such as UDP.

In Ethernet networks, a unique hardware address identifies each interface on the network. IP addresses are more flexible because they aren't specific to a network type. A message that uses IP can travel through different types of networks, including Ethernet, token-ring, and wireless networks, as long as all of the networks support IP.

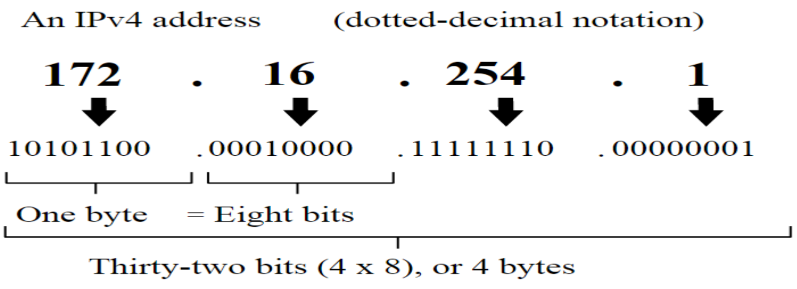


Figure 40 - IP address format

In sending a message, the TCP layer passes the TCP segment and the source and destination addresses to the IP layer. The IP layer encapsulates the TCP segment in an IP datagram, which consists of a header followed by a data portion that may contain a UDP datagram or a TCP segment.

The header has fields for the source and destination IP addresses, error checking of the header, routing, and a value that identifies the protocol, such as TCP or UDP, used by the data portion.

In a similar way, a UDP layer may pass a UDP datagram to the IP layer. In receiving a message, the IP layer receives an IP datagram from a lower level in the network stack. The IP layer performs error-checking and uses the protocol value to determine where to pass the contents of the data portion.
In the IP header, the source and destination IP addresses identify the sending and receiving computers. Each computer in a network that uses IP addresses must have an address that is unique within the network or networks that the sending computer can communicate with. Local networks can use addresses in three blocks reserved for local networks. A computer that communicates over the Internet must have an address that is different from the address of every other computer on the Internet. The Internet Corporation for Assigned Names and Numbers (ICANN) assigns blocks of addresses to Internet Service Providers and others who may in turn assign portions of their addresses to other users.
Three protocols often used along with IP for assigning and learning IP addresses are the dynamic host configuration protocol (DHCP), the domain name system (DNS) protocol, and the Address Resolution Protocol (ARP).

• 3.3.5 Summary

Any web server has only one role, to implement the HTTP protocol. This protocol allows a browser to initiate a request which a server will respond to it. But today's web services has high resolution graphics and streaming media, yet surprisingly, HTTP is a very simple protocol. All it does is to send a request and receive a respond.

HTTP is a stateless protocol, there is no intelligence in the operation: there is no decision-making or context, and there are no scripts or code to execute.

HTTP understands only nine operations called methods, of which the most important are GET and POST. GET is used to download a "resource" located at a specified location (the "uniform resource locator", or URL). The response to a GET request contains the resource, accompanied by HTTP header information as shown in figure 41. When someone clicks a link in a web browser to go to a new page, they are sending a GET request to the server asking for the HTML page located at the URL they clicked.

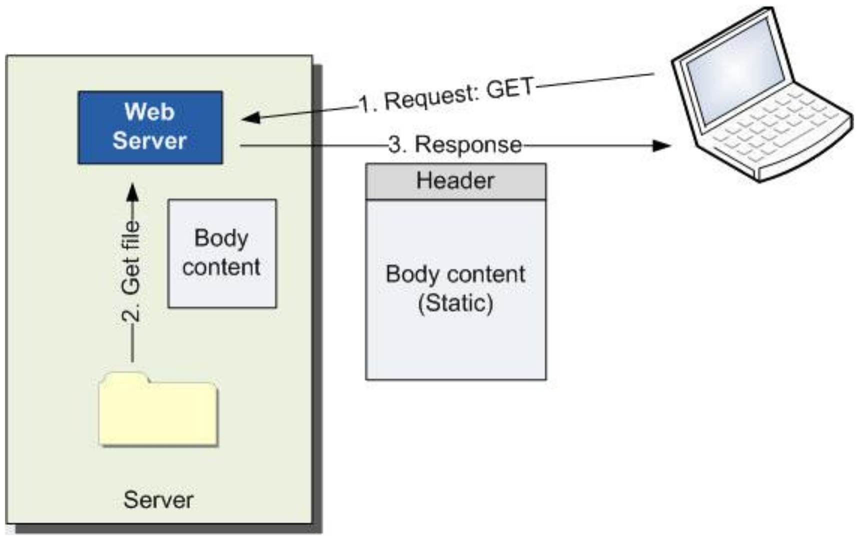


Figure 41 - A GET request simply results in a static file being returned.

POST is used to submit data from the web browser back to the server. This is the request that is generated when someone hits the "Submit" button on a filled-out form. A basic web server cannot process a POST request, since it will have no idea what to do with the data in the request. It must rely on some other program or utility to process the data.

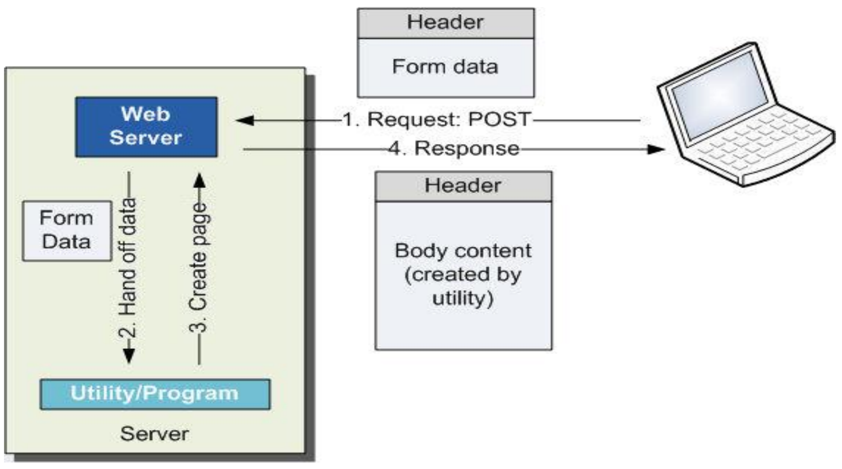


Figure 42 - POST requests must be handed off to some other utility that will determine an appropriate response.

Most web-based applications need much more than this. In particular, embedded web applications generally must be able to let the web functionality affect the device operation - the remote browser has to be able to "talk" to the system. A web server has no such capability, so everything beyond the simple processing of HTTP must be written as a part of system development.

The low-level code that will interact directly with the system hardware will likely be written in C, and, because these routines are so system-specific, they will need to be written during development no matter how the web connectivity is implemented. The challenge is that a web server has no way of accessing those routines, meaning that some way of connecting the two must be built.

The two high-level elements that must be created are:

- A way for the web browser to access some other program;
- A program that can take an HTTP request and, from the request contents, invoke the appropriate C code (Figure 43).

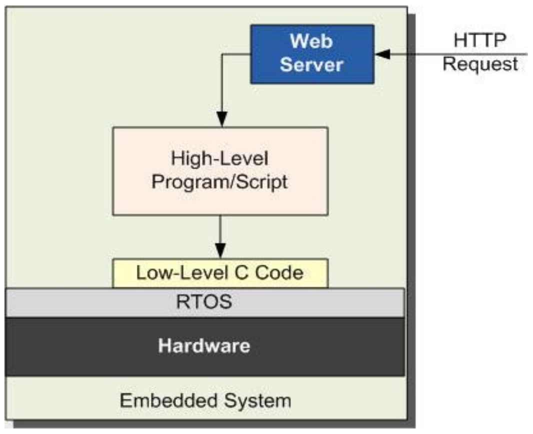


Figure 43 - An environment is needed to allow the web server to hand off to a high-level script, which accesses low-level C routines

▪ 3.4 Mobile Application

• 3.4.1 Smartphones

At one time, phones were used entirely to support voice communication, conversing with other people and reduce the gap between them, while computers run software applications. As the prosperity of the technology nowadays computer has become a small chip that can be existed in a mobile phone to integrate what is called smartphone. People are beginning to replace their personal computers with smartphones.

Smartphones have become more and more central to our daily life. The invention of the smartphone is a turning point in history because it's a huge addition of technology. Without smartphones, there would not be such thing as using internet on the go.

A smartphone is a device that includes many functions similar to those operated on the computer. Smartphones provides cell phone with a hand-held computer, camera and digital video processing, GPS, media player, touchscreen user interface, messaging, e-mail capability, large data storage to add any software applications, internet access through wireless connection, so this is useful for quick references like using maps in the car to find the destination. This is the reason why smartphones are a great addition to technology.

As smartphones get smarter by using new brilliance technology in the phone and the cloud, they'll begin to comprehend our life patterns, reason about our health, help us explore as the day progressed, and intercede on our behalf.

• 3.4.2 Mobile Platforms

3.4.2.1 Android

Android is a mobile operating system made by American company; Google. The Android platform is based on the Linux kernel system. It most commonly comes installed on a variety of touch screen devices like smartphones and tablets from a host of manufacturers offering users access to Google's own services like Search, YouTube, Maps, Gmail and more.

Android is developed in private by Google until the latest changes and updates are ready to be released, at which point the source code is made available publicly.

Figure 44 - Android Logo

Regarding to Android applications (apps), it provides many applications on different matters of life on its store. As Android is open-sourced, it supports anyone to make their own application using the Android software development kit (SDK) and, often, the Java programming language that has complete access to the Android Application Programming Interface (API).Java may be combined with C/C++, together with a choice of non-default runtimes that allow better C++ support.

3.4.2.2 IOS

IOS stands for iPhone Operating System. It is an operating system developed by Apple and used in its products like iPhone, iPod and iPad. It distributed exclusively for Apple hardware.

It was first developed in 2007 and major updates are released on a yearly basis. Its biggest competitor is Android operated smartphones.

Figure 45 - Apple Logo

3.4.2.3 Windows phone

Windows phone is a mobile operating system for smartphones and mobile devices that serves as the successor to Microsoft's initial mobile OS platform system, Windows Mobile.

Figure 46 - Windows Phone Logo

3.4.2.4 Blackberry

Research In Motion (RIM) released its first BlackBerry device in 1999, providing secure real-time push-email communications on wireless devices. Services such as BlackBerry Messenger provide the integration of all communications into a single inbox. A number of corporations have adopted the BlackBerry to remain in constant communication with their employees and clients.

Figure 47 - Blackberry Logo

• 3.4.3 OS Market Share

The overall smartphone business commenced the year with a normal post-occasion scattering in shipment volume contrasted with the past quarter's remarkable results. As indicated by information from the International Data Corporation (IDC) Worldwide Quarterly Mobile Phone Tracker,

Vendors shipped a total of 334.4 million smartphones worldwide in the first quarter of 2015, up 16.0% from the 288.3 million units in first quarter in 2014 but down by 11.4% from the 377.6 million units shipped in fourth quarter in 2014.

Android dominated the market with a 78.0% share in first quarter in 2015. Samsung reasserted its global lead with a renewed focus on lower-cost smartphones.

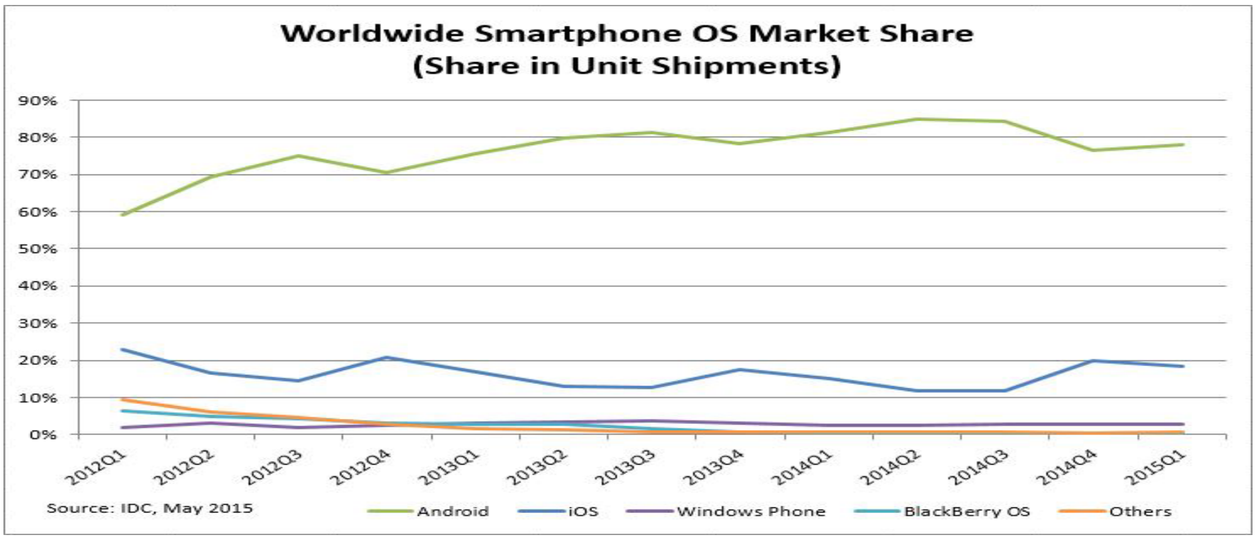


Figure 48 - Mobile market share within last 4 years according to International Data Corporation (IDC), May 2015

Period	android			BlackBerry	Other OS
Q1 2015	78.0%	18.3%	2.7%	0.3%	0.7%
Q1 2014	81.2%	15.2%	2.5%	0.5%	0.7%
Q1 2013	75.5%	16.9%	3.2%	2.9%	1.5%
Q1 2012	59.2%	22.9%	2.0%	6.3%	9.5%

Table 18 - Mobile market share within last 4 years

CHAPTER 4

Project Description & Related Work

- **4.1 Related Work**

- **4.1.1 Automotive Mobile Platform**

Automotive Mobile Platform is a graduation project made by 6 students in Ain Shams University Faculty of Engineering, Computer and Systems department, and sponsored by Valeo.

Automotive Mobile Platform is a mobile application that sends SMS messages to control some ECUs in the vehicle e.g. open and close lights of car, open the trunk. They used GSM board and Arduino board to communication between the mobile and the ECUs in the vehicle. The GSM board have the GSM protocol which provide the necessary AT commands to send and receive messages .The Arduino board have a UART driver in its AVR microcontroller to make a UART interface which have the responsibility for communicating between the GSM board and the ECUs in the CAN bus to send and receive SMS messages from GSM board to ECUs or vice versa.

They made GUI interface to test the project which have 4 LEDs that represent Light, Engine, Windows, and Doors as shown in Figure 49.

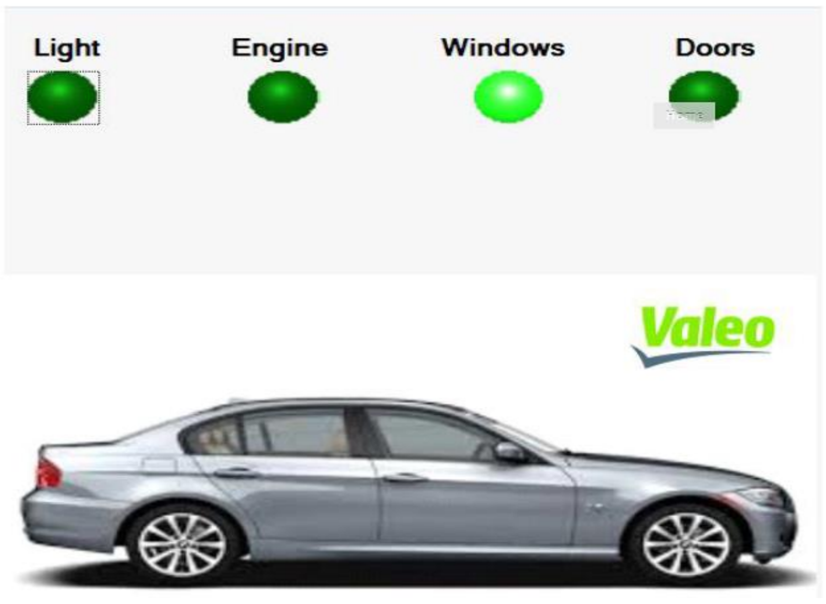


Figure 49 - Automotive Mobile Platform Ein Shams team Graduation project

Comparing Automotive Mobile Platform with App Your Car, they didn't interface with a CAN bus in a real car .They just made a simulation test on a GUI interface for CAN bus that turn on LEDS instead of controlling the corresponding ECUs e.g. Light, Engine, Windows, and Doors. In our project, we use a CAN bus shield and OBD-II cable to interface with CAN bus in a real car. App Your Car application characterize a real communication with real car.

In addition, the cost of the GSM board is much greater than the Ethernet shield that we used in our project to send the data that we monitored from the vehicle to our Android App throw the internet.

• 4.1.2 Auto Performance Analyzer

Auto Performance Analyzer is a graduation project made by Texas University, Faculty of Engineering, Computer and science department. [21]

Auto Performance Analyzer mobile application provides a digital screen for all sensors in the vehicle and interface with the CAN bus communication system installed in every vehicle since 1996. This app will give users the ability to check and clear error codes, display real time data about the vehicle's performance, logging database and record different parameters of their deciding to be demonstrated in a graphical representation. They used STN1170 Bluetooth OBD-II Adapter that have microcontroller to interface with CAN bus in the vehicle and Bluetooth module to send real time data back to the mobile app by Bluetooth connection as shown in figure 50.

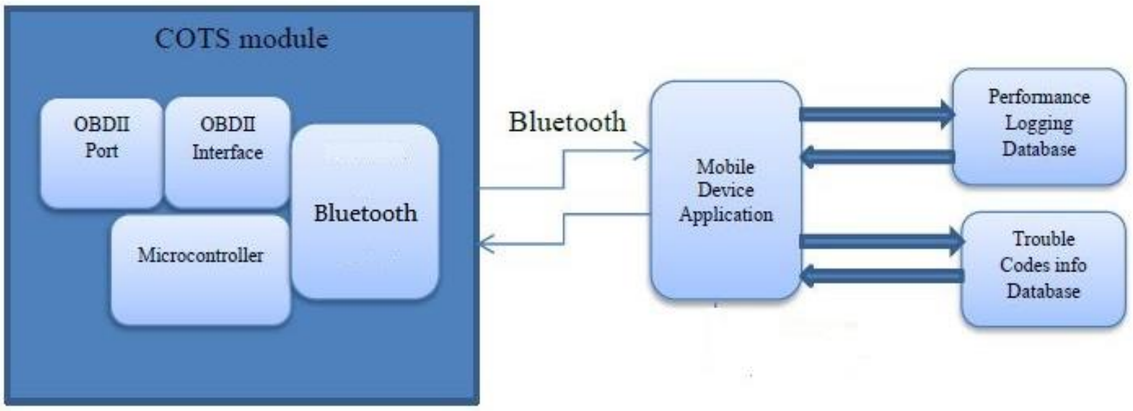

COTS module
OBDII Port
OBDII Interface
Bluetooth
Microcontroller
Bluetooth
Mobile Device Application
Performance Logging Database
Trouble Codes info Database

Figure 50 - Texas university graduation project

Comparing Auto Performance Analyzer with App Your Car, they used a Bluetooth module to send the data from the vehicle to the app .The maximum range of the Bluetooth connection is 10 m so any user will not be able to monitor his/her vehicle from any place, but in App your Car project we used Ethernet shield that allows the user to show all monitor parameters on our Android App from any place throw the internet. [21]

▪ 4.2 Project Definition

A smart mobile platform used to standardize the Vehicle-Mobile interface. App Your Car is the smartphone gear specifically designed to give you an easy and safe way to use your phone or tablet in your car. You can use our android application to monitor some functions in your car through the internet such as vehicle speed, vehicle RPM, the coolant temperature of the engine, the throttle percentage of the gas pedal … etc. Moreover, our app can diagnose all the faults in your car with one click and sends back a full report about all the troubles in the vehicle.

▪ 4.3 Project Flow Description

By connecting our integrated kits (Arduino UNO, CAN bus and Ethernet shields) to the OBD-II pinout in the vehicle and selecting mode 2 from OBD-II modes, we can have the full access to all the information of the ECUs of the vehicle through the CAN bus. After starting the engine, all the data requested from the OBD-II interface are available on an array and ready to be read or displayed. So it has been sent over the internet to be displayed on the android app which is also connected to the Internet.

By selecting mode 3 from OBD-II modes to get the DTC (Diagnosis Trouble Codes), the received 6 digits from the CAN frame are decoded to represent a certain trouble, if we receive a CAN message full of zeros that means that no trouble code has been detected. After diagnosing the car, if it has troubles, the corresponding DTC will be sent to the android application over the internet. If no troubles have been detected, you will get a message informs you that your car is clean with zero DTC

▪ 4.4 How to Select a Microcontroller

• CPU Architectures

Basically, there are two processor architectures according to memory accessing, Von-Neumann and Harvard architectures.

Regarding to Von Neumann architecture, it has a single and common memory space where data and program instructions are stored. There is a single data bus shared between them which fetches both data and instructions. Each time the CPU fetches a program instruction it may have to perform one or more read/write operations, it must wait until these subsequent operations are completed before it can fetch the next instructions.

While Harvard architecture has separate memory areas for data and program instructions. There are two buses, one connects CPU to data memory and the other one connects CPU to program memory. The CPU can read an instruction and perform a data memory access at the same time. This speeds up program execution time.

Also there are two processor architectures according to number of instructions, RISC (Reduced Instruction Set Computing) and CISC (Complex Instruction Set Computing). RISC is a CPU design that recognizes only a limited number of instructions. Instructions simple and executed quickly. RISC aims to optimize execution of instructions by limiting the capabilities of a single instruction, thus gaining speed from execution point of view but code size will be large for a dedicated function.

While CISC is a CPU design that recognizes a large number of instructions. Instructions are complex and take large execution time for each one. CISC aims to integrate several functionalities in one instruction, in order to limit the program size and limit memory access in order to gain some execution speed.

• Microcontrollers

Microcontrollers are referred to single chip devices or single chip computers in a small size that its resources are more limited than those of a desktop personal computer. Microcontroller consists of a central processing unit, data memory, program memory and some peripherals (I/Os, timers, interrupts, ADC, serial communication). Microcontrollers are designed for standalone operation. They include a processing unit of 8-bit, 16-bit or 32-bit.
Microcontrollers most often use Harvard and RISC architectures. Microcontrollers are considered a System On Chip (SOC). Due to advancements in silicon technologies, it is possible to add more functionalities to the processor system on chip which has multiple functional units on one piece of silicon, that can be considered integrating all components of a computer or other electronic systems into a single integrated circuit.

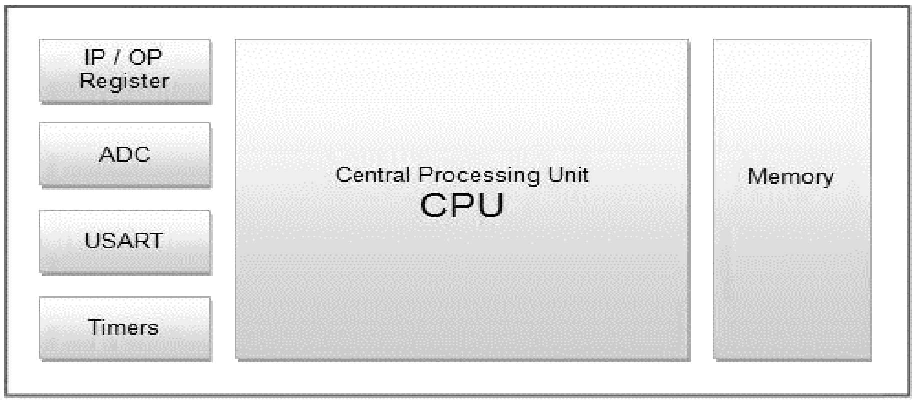


Figure 51 - Architecture of AVR microcontroller

- **AVR**

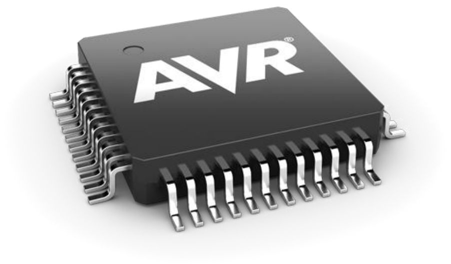


Figure 52 - AVR

An AVR is a type of microcontroller devices that was developed in the year 1996 and manufactured by Atmel Corporation. The architecture of AVR was conceived by two students at the Norwegian Institute of Technology (NTH). AVR derives its name from its developers and stands for Alf-Egil Bogen and Vegard Wollan RISC microcontroller. AVR is a modified Harvard architecture RISC single chip microcontroller.

There are three families available of AVR microcontrollers

- TinyAVR: Less memory, small size, suitable only for small projects.
- MegaAVR: These are the most popular family having good amount of memory up to 256 KB, higher number of peripherals and suitable for operate to complex applications.
- XmegaAVR: Used commercially for complex applications, which require large program memory and high speed.

A lot of things are needed when using AVR microcontrollers in any application or a project. First regarding hardware components, As the AVR microcontroller is just an integrated circuit (IC), so a bread board is needed to fix the IC on it, a battery to power the chip and in order to upload the code onto the program memory of the chip an external hardware circuit called burner is also needed, so programmers have to use this external hardware when uploading their codes to see project performance and this is an annoyable factor on AVR controllers.

Second, for software components AVR software is commonly developed in assembly or C and loaded onto the chip. Atmel Corporation provides different versions of tool chains for windows or Linux to be used in AVR projects based called Atmel studio.

• Arduino

Arduino is an open source computing prototyping platform based on a simple AVR microcontroller, it can be used to develop interactive objects, taking inputs from a variety of switches, sensors or actuators, and controlling a variety of lights, motors, and other physical outputs. Arduino projects can be stand-alone, or they can communicate with software running on your computer.

Figure 53 – Arduino Logo

Arduino provides many types of boards with different number of input and output pins, timers, UART, Interrupts. For examples Arduino Uno, Leonardo and Mega. Arduino also provides many types of shields that can be placed on Arduino board and work together. For example Ethernet shield, WIFI and GSM.

For using an Arduino board in a project or application, from the hardware point of view as the Arduino is not only an IC chip, there is no need for using a bread board to build the hardware but by using connectors to interact with the I/O pins. Also an advantage of using Arduino board that there is no need for external hardware circuit that called programmer or burner to upload the code on to the board but Arduino provides a USB port with USB cable to connect the board with the computer easily.

From the software point of view the software is developed in a C-like language. The Arduino software runs on Windows, Macintosh OSX, and Linux operating systems. Arduino IDE (Integrated Development Environment) uses a simplified version of C++, making it easier to learn to program. Arduino also provides a standard form factor that breaks out the functions of the micro-controller into a more accessible package.

Additionally, the Arduino software is published as open source tools, available for extension by programmers. The language can be expanded through C++ libraries, and for those who wants to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based.

• Parameters of Selecting a Microcontroller

The hardware and software engineers should work out the high levels of the system, block diagram and flowchart, before any thought about the microcontroller. And then there is enough information to start making a rational decision on microcontroller selection.

Selecting the appropriate microcontroller for a project can be a daunting task. Not just there are various technical components to consider, there are likewise business case, for example, cost and lead-times that can cripple a project. At the start of a project there is a freedom to jump in and start selecting a microcontroller before the details of the system has been hashed out.

There are easy points should be taken in mind to ensure that the right choice is made:

- Number of I/O ports.

- Peripherals required (Timers, ADC, USART, SPI and EEPROM).
- Memory requirements. Maximum clock speed.
- Real-Time considerations.
- Number of interrupts required
- Power Consumption / sleep modes.
- Availability and Cost.
- Vendor Support.
- Development Environment.

4.5 Components Used

- **Arduino Uno**

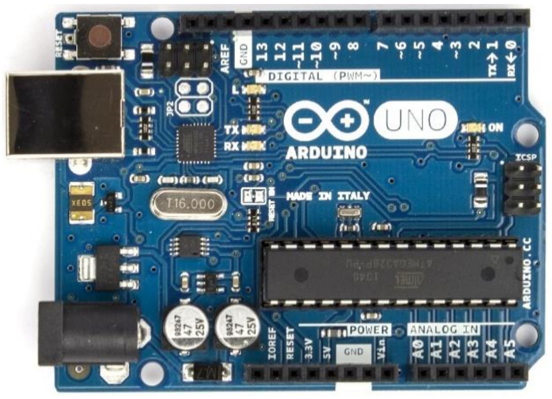


Figure 55 - Arduino Uno R3 Front

Figure 54 - Arduino Uno R3 Back

Description:

The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller

Features:

- Microcontroller: ATmega328
- Operating Voltage: 5V
- Input Voltage: 7-12V
- Digital I/O Pins: 14 (of which 6 provide PWM output)
- Analog Input Pins: 6
- DC Current per I/O Pin: 40 mA
- DC Current for 3.3V Pin: 50 mA
- Flash Memory: 32 KB (ATmega328) of which 0.5 KB used by bootloader
- SRAM: 2 KB
- EEPROM: 1 KB
- Clock Speed: 16 MHz

- **Spark Fun CAN Bus Shield**

Description:

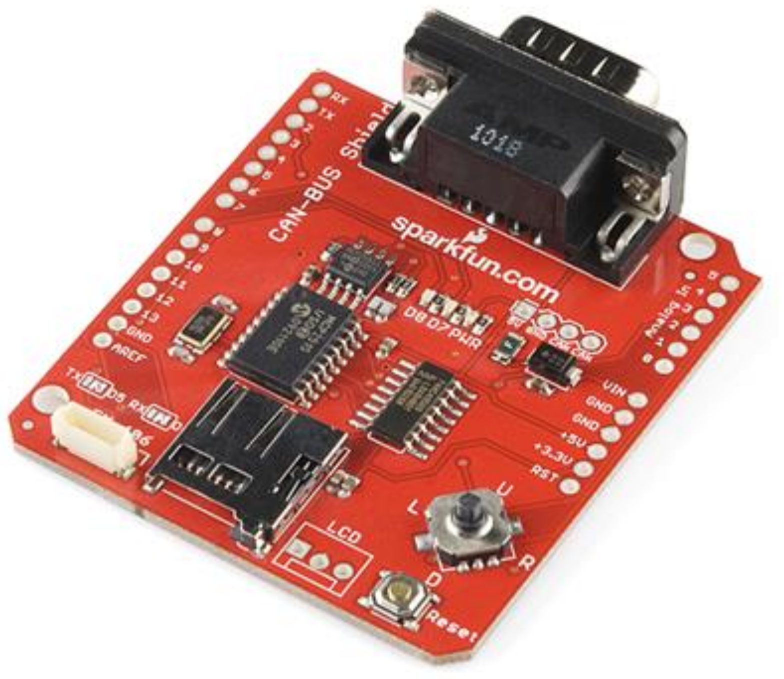


Figure 56 - Spark Fun CAN bus sheild

This shield gives the Arduino CAN-Bus capability. It uses the Microchip MCP2515 CAN controller with MCP2551 CAN transceiver. CAN connection is via a standard 9-way sub-D for use with OBD-II cable. Ideal for automotive CAN application. The shield also has a micro SD card holder, serial LCD connector and connector for an EM406 GPS module. Making this shield ideal for data logging application.

Features:

- CAN v2.0B up to 1 Mb/s
- High speed SPI Interface (10 MHz)
- Standard and extended data and remote frames
- CAN connection via standard 9-way sub-D connector
- Power can supply to Arduino by sub-D via resettable fuse and reverse polarity protection.
- Socket for EM406 GPS module
- Micro SD card holder
- Connector for serial LCD
- Reset button
- Joystick control menu navigation control
- Two LED indicator

- **OBD-II to DB9 Cable**

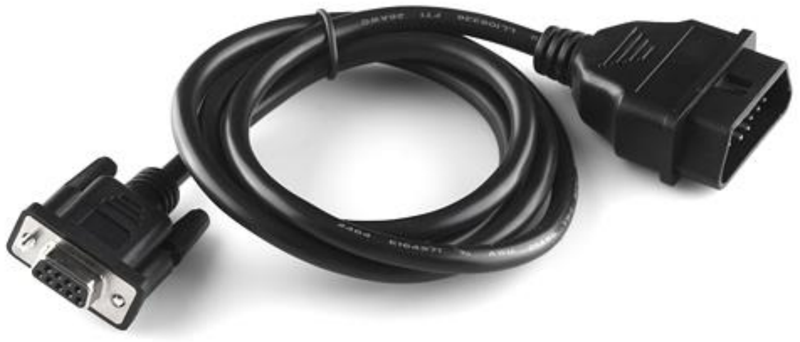


Figure 57 - OBD-II to DB9 Cable

Description:

This cable allows you to access the pins on your car's OBD-II connector. It has an OBD-II connector on one end which is connected to the car and a DB9 female serial connector on the other which is connected to the CAN bus shield. The overall length of the cable is 5 feet.

Pin Description	OBDII	DB9
J1850 BUS+	2	7
Chassis Ground	4	2
Signal Ground	5	1
CAN High J-2284	6	3
ISO 9141-2 K Line	7	4
J1850 BUS-	10	6
CAN Low J-2284	14	5
ISO 9141-2 L Line	15	8
Battery Power	16	9

Table 19 - Pin describtion of OBD-II to DB9 Cable

- **Arduino Ethernet Shield**

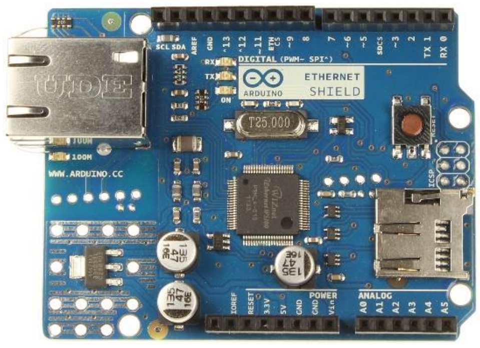


Figure 58 - Ethernet Shield Front

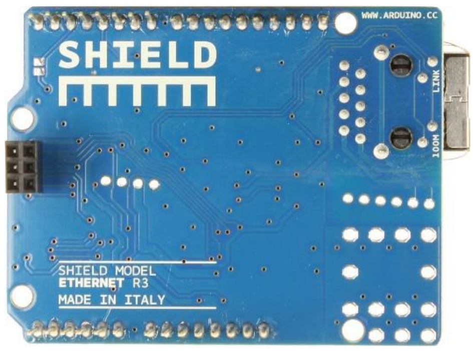


Figure 59 - Ethernet Shield Back

Description:

The Arduino Ethernet Shield allows an Arduino board to connect to the internet. It is based on the Wiznet W5100 Ethernet chip. The Wiznet W5100 provides a network (IP) stack capable of both TCP and UDP. It supports up to four simultaneous socket connections. Using the Ethernet library to write sketches which connect to the internet using the shield. The Ethernet shield connects to an Arduino board using long wire-wrap headers which extend through the shield. This keeps the pin layout intact and allows another shield to be stacked on top.

The Ethernet Shield has a standard RJ-45 connection, with an integrated line transformer and Power over Ethernet enabled.

Features:

- IEEE802.3af compliant
- Low output ripple and noise (100mVpp)
- Input voltage range 36V to 57V
- Overload and short-circuit protection
- 9V Output
- High efficiency DC/DC converter: type 75% @ 50% load
- 1500V isolation (input to output)

- **Portable Battery Powered 3G/3.75G Wireless/Ethernet Router**

Figure 60 - TP Link Portable Battery Powered 3G/3.75G Wireless/Ethernet Router

Description:

Portable router that provides wired and wireless Internet connection by using a 3G USB modem.

Features:

- Share a 3G/4G mobile connection, compatible with LTE/ HSPA+/ HSPA/ UMTS/ EVDO 3G/4G USB modems
- Battery powered, portable design, light and small enough for you to take anywhere
- Wireless N speed up to 150Mbps
- IP-based bandwidth control allows administrators to determine how much bandwidth is allotted to each connected device
- A micro USB port to be connected to your laptop or power adapter for power supply
- Three working modes: 3G/4G Router, Travel Router (AP), WISP Client Router
- Easy operation modes change at a flip of the operation mode switch

▪ 4.4 Tasks Description

After analysis of the project, we divided it into four main parts:

A) Arduino Ethernet Web Server

Where the IP datagrams is sent and received between the server (Arduino kit) and the client (Android application) using HTTP request and response methods and that will be illustrated in details in the next chapter.

B) Reading Real Time Values from Different ECUs in the Vehicle

By attaching a CAN bus shield to the Arduino Uno board, and using the OBD-II to DB9 cable to connect the board to the vehicle. Now we can read all the real time reading using mode 1 by initiating a request to CAN bus using CAN ID 7df. After receiving the response and using table 7 we can read all the PIDs which is supported by the car.

C) Reading the DTCs of the Vehicle

As we stated before in mode 3 in the on board diagnostic section in chapter 3, a request for this mode returns a list of the Diagnosis Trouble Codes (DTCs) that have been set. The list is defined using the ISO 15765-2 protocol. Using the same procedure of requesting a PID by sending a CAN message with CAN ID 7df but this time we are using mode 3, so the reply will be different and for more information you can go back to mode 3 in OBD section in chapter 3 you will find tables used for decoding these DTCs.

D) Mobile Application

By using Android Studio, the official IDE for developing on the Android platform, we developed an Android application which can serve all our needs and review all the readings from the ECUs in task B and the DTCs and their number in task C and also it can be connected to the internet to achieve task A.

It is very clear how our project is modular as it allows typical applications to be divided into modules, as well as integration with similar modules, which helps any further enhancements in the future to take place easily and also in testing procedures we can test each part separately to perfectly debug any errors in the code,

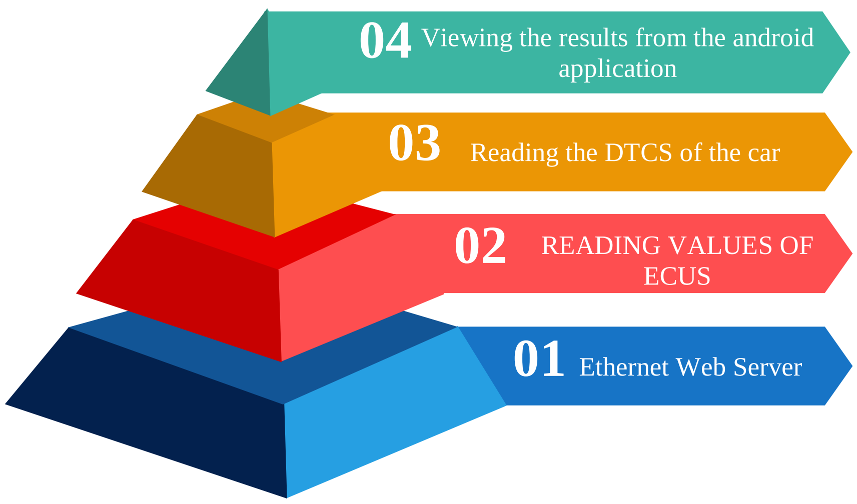

04 Viewing the results from the android application
03 Reading the DTCS of the car
02 READING VALUES OF ECUS
01 Ethernet Web Server

Figure 61 - The representation of the tasks of our project in an origami pyramid

CHAPTER 5

Technical Approach

"It always seems impossible until it's done."
Attributed to ***Nelson Mandela***

From figure 55 in the end of chapter 4, it is clear that we have 4 main tasks. So in this chapter these 4 tasks will be implemented individually then we will integrate them together in the last section of this chapter to get our final output.

Implementing each task alone will give us the potential to test our project and to reach our final output efficiently without any errors.

All the complete codes of these tasks are found in APPENDIX II.

▪ 5.1 Arduino Ethernet Web Server

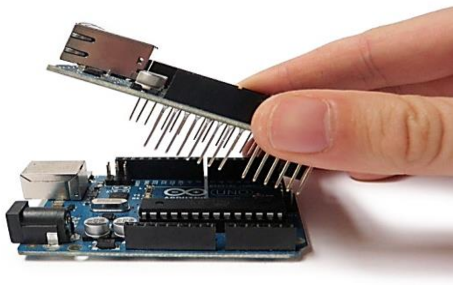


Figure 62 - Ethernet Shield attached to Arduino Uno board

Our shield is Arduino compatible, so first we have to include the Ethernet.h and SPI.h libraries which is supported by the Arduino IDE.

Any Arduino code is divided into 2 main functions, `void setup ()` and `void loop ()`. The `void setup ()` is the setup routine which runs once when the Arduino run and is the portion of code which is used to initialize variables, pin modes, start using libraries, etc.

After creating a `setup ()` function, the `loop ()` function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

First of all we have to obtain the IP address of the Ethernet shield which is auto assigned through Dynamic Host Configuration Protocol (DHCP) in examples of the Ethernet library in the Arduino IDE. (Called DHCP Address printer)

```
IPAddress ip (192, 168, 1, 171); // The IP address is dependent on your Network
```

After obtaining an IP address we have to initialize the server to listen for incoming connections on a specified port, which will be port 80 in our case because it is the default port for HTTP traffic.

```
EthernetServer server (80);
```

In the `void setup ()` function, we will open a serial communications with baud rate 9600 to see the request and response methods transmitting between the server and the client.

```
Serial.begin (9600);
```

Then, initializing the Ethernet library and the network settings with the obtained IP address and the MAC address of the shield.

```
Ethernet.begin (mac, ip);
```

The MAC (Media access control) address for the device (array of 6 bytes). This is the Ethernet hardware address of your shield. Newer Arduino Ethernet Shields include a sticker with the device's MAC address. For older shields, choose your own

Furthermore, telling the server to listen for the upcoming connections on the obtained IP address and the MAC address of the shield.

```
server.begin ();
```

In the `void loop ()` function, we create a client which can connect to a specified IP address and port by getting a new client that is connected to the server and has data available for reading. The connection persists when the returned client object goes out of scope.

```
EthernetClient client = server.available ();
```

By checking if the specified Ethernet client is ready or not. If the client is ready we will enter the if loop and continue in our code.

```
if (client) {
.........
}
```

There are two more checks to make sure that they are true before establishing the connection between the server and the client.

```
while (client.connected ())
if (client.available ())
```

The while loop is set to be true after describing Whether or not the client is connected. The if loop will Returns the number of bytes available for reading (that is, the amount of data that has been written to the client by the server it is connected to).

After all these conditions set to be true, now we can begin to read the request from client to server. By reading the next byte received from the client connected to the server, and saving it in a character c, then writing c on the serial monitor, this data is sent as a byte or series of bytes.

```
char c = client.read();
Serial.write(c);
```

If you've gotten to the end of the line (received a newline character) and the line is blank, the HTTP request has ended, so you can send a reply. So it will be the interior if loop condition.

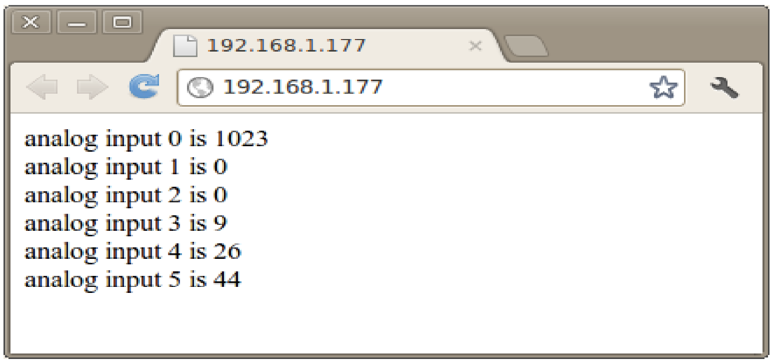
```
if (c == '\n' && currentLineIsBlank)
```

Now we can begin to send a standard HTTP response header.

```
client.println ("HTTP/1.1 200 OK");
client.println ("Content-Type: text/html");
client.println ("Connection: close");
client.println ("Refresh: 5");
client.println ();
client.println ("<!DOCTYPE HTML>");
client.println ("<html>");
```

In this portion of code, we can add any HTML code to open a web page with the obtained IP address of the Ethernet shield. But for testing, we will use a simple HTML code to send the values of the analog input pins of the Arduino board then we will `break` from the code.

```
for (int analogChannel = 0; analogChannel < 6; analogChannel++) {
            int sensorReading = analogRead (analogChannel);
            client.print ("analog input ");
            client.print (analogChannel);
            client.print (" is ");
            client.print (sensorReading);
            client.println ("<br />");
          }
          client.println ("</html>");
          break;
```



analog input 0 is 1023
analog input 1 is 0
analog input 2 is 0
analog input 3 is 9
analog input 4 is 26
analog input 5 is 44

Figure 63 - The HTML web page which is viewed in the browser

Finally, after the request and response is done completely, the client will be disconnected from the server.

```
client.stop ();
```

- **5.2 Reading Real Time Data from Different ECUs in the Vehicle**

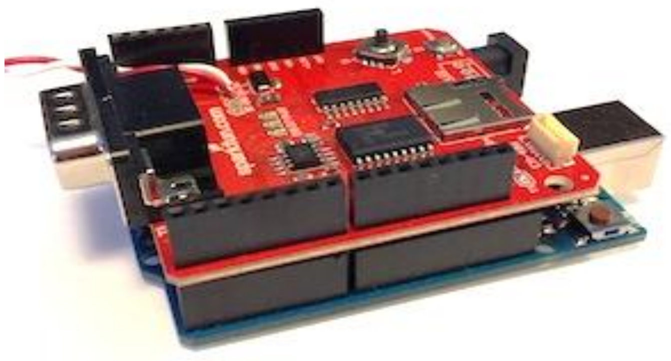


Figure 64 - CAN bus shield attached to Arduino Uno board

As we stated before in chapter 4, one of the main automotive main building units is the on board diagnostics system. We knew that to use this interface we have to know well all its modes and how to extract data using the PIDs of each ECU inside the car.

- **5.2.1 Supported Parameters**

We also previously said in our book, not all the PIDs are supported in all cars. Consequently, we have to test whether the PID is supported in the vehicle we are testing our code on or not.

By referring to the PID full table in APPENDIX I we can make this brief table of PIDs that we will use in this test.

	PID (hex)	Data Bytes Returned	Description
Mode 1	00	4	PIDs supported [01 - 20]
	20	4	PIDs supported [21 - 40]
	40	4	PIDs supported [41 - 60]
	60	4	PIDs supported [61 - 80]
	80	4	PIDs supported [81 - A0]

Table 20 - PID supported

First of all we have to initialize the CAN bus shield by the following functions which are implemented in a library included to the file called <AppYourCar.h>

```
obd.setSlow(false);
obd.setExtended(false);
obd.setDebug(false);
obd.begin();
```

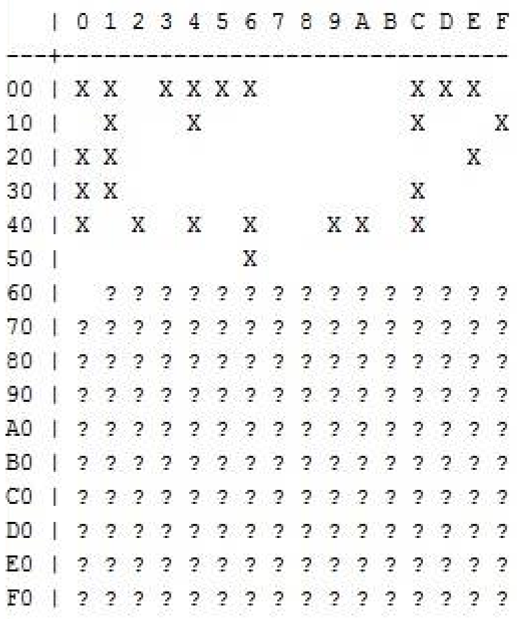
Then we will print a table on the serial monitor to make it easy to know which of the PIDs are supported by this vehicle

```
Serial.println ("   | 0 1 2 3 4 5 6 7 8 9 A B C D E F");
Serial.println ("---+---------------------------------");
```

Then we will write an if statement nested in for loop nested in another for loop to loop on the 4 PIDs (00,20,40,60) and write out on the serial monitor a table with all the supported PIDs.

```
for (int i = 0; i <= 0xf0; i += 0x10) {
  Serial.print (i, HEX);
  Serial.print (i == 0 ? "0 |" : " |");
  for (int j = 0; j <= 0x0f; j++) {
    boolean supported;
    if (obd.isPidSupported (i + j, supported)) {
      Serial.print (supported ? " X" : "  ");
    } else {
      Serial.print (" ?");
    }
    delay (50);
  }
  Serial.println ();
}
```

The following figure is the serial print of the output of the supported parameters, this output will differs from one vehicle to another depending on the manufacturer of the vehicle.



```
   | 0 1 2 3 4 5 6 7 8 9 A B C D E F
---+--------------------------------
00 | X X   X X X X           X X X
10 |   X     X               X     X
20 | X X                         X
30 | X X                     X
40 | X   X   X   X     X X   X
50 |             X
60 |   ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
70 | ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
80 | ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
90 | ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
A0 | ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
B0 | ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
C0 | ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
D0 | ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
E0 | ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
F0 | ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
```

Figure 65 - Supported Parameters

"X" : means that this PID exists and can be read through OBD-II cable.

" " or "?" : means that this PID is not supported.

- **5.2.2 Reading the values of the supported Parameters**

After we knew what the supported PIDs are, we can easily read these PIDs as we stated before. Using the function which is implemented in the `AppYourCar.h` library `obd.getPidAsInteger` or `obd.getPidAsFloat` we can get back the reading of the required ECU by adding its PID in any one of these two functions depending on whether I want to read a float value or I don't care with the floating point then I will read an integer value.

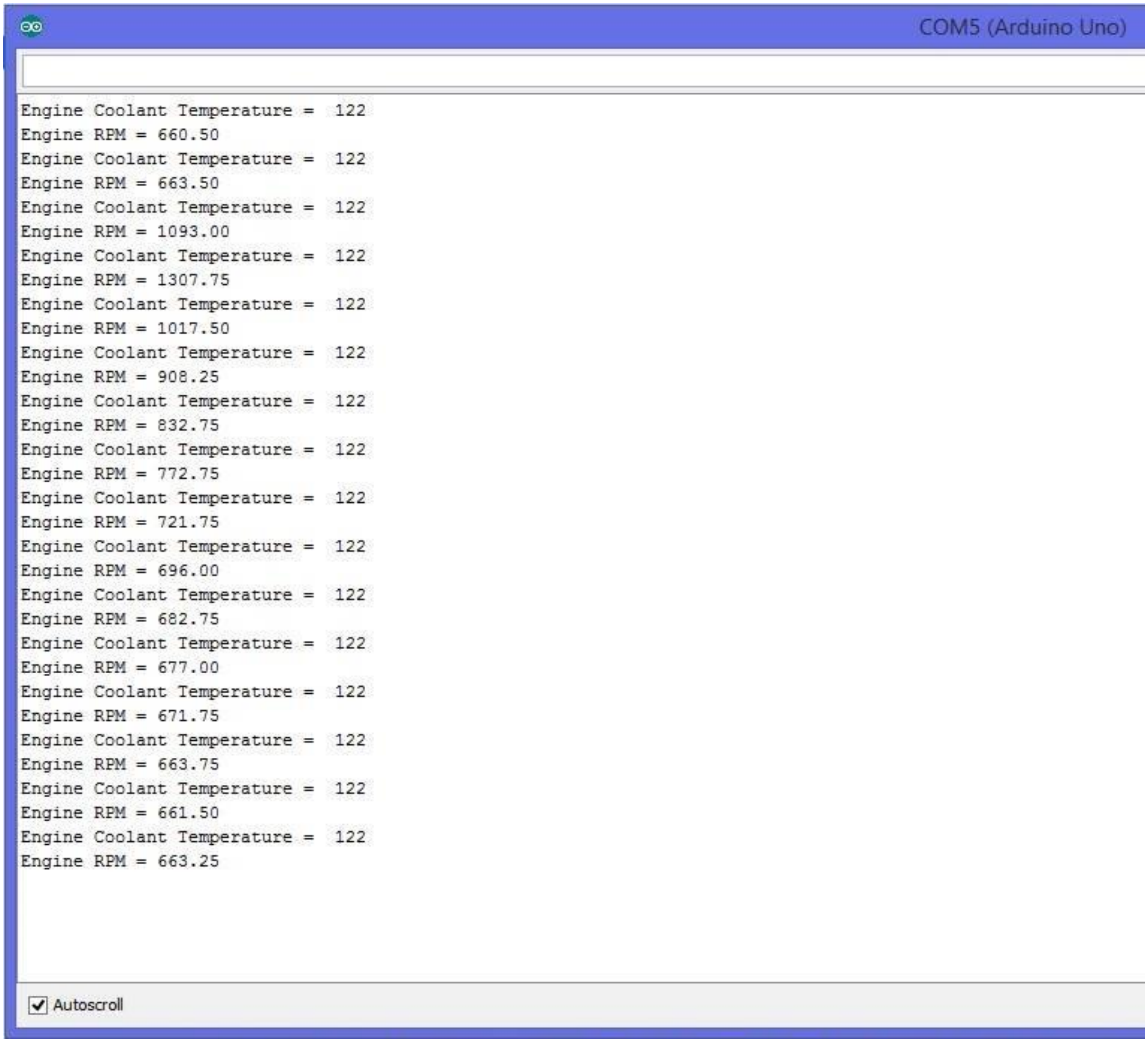
Likewise 5.2.1, we will initialize the shield with the same 4 functions.

Referring to table 7 in chapter 3, there you will find the most important PIDs that we are willing to read. Specifically you will find all the PIDs in APPENDIX I.

For example, the engine coolant temperature has PID = 0x05.

```
unsigned int temp;
float rpm;
obd.getPidAsInteger (0x05, temp);
obd.getPidAsFloat (0x0c, 0.0f, 16383.75f, rpm);
Serial.print (F ("Engine Coolant Temperature =  "));
Serial.println (temp);
Serial.print (F ("Engine RPM = "));
Serial.println (rpm);
```

Accordingly, the value of the engine coolant temperature will be written on the serial monitor as shown in figure 67.



```
Engine Coolant Temperature =  122
Engine RPM = 660.50
Engine Coolant Temperature =  122
Engine RPM = 663.50
Engine Coolant Temperature =  122
Engine RPM = 1093.00
Engine Coolant Temperature =  122
Engine RPM = 1307.75
Engine Coolant Temperature =  122
Engine RPM = 1017.50
Engine Coolant Temperature =  122
Engine RPM = 908.25
Engine Coolant Temperature =  122
Engine RPM = 832.75
Engine Coolant Temperature =  122
Engine RPM = 772.75
Engine Coolant Temperature =  122
Engine RPM = 721.75
Engine Coolant Temperature =  122
Engine RPM = 696.00
Engine Coolant Temperature =  122
Engine RPM = 682.75
Engine Coolant Temperature =  122
Engine RPM = 677.00
Engine Coolant Temperature =  122
Engine RPM = 671.75
Engine Coolant Temperature =  122
Engine RPM = 663.75
Engine Coolant Temperature =  122
Engine RPM = 661.50
Engine Coolant Temperature =  122
Engine RPM = 663.25
```

Figure 66 - Serial monitor of Engine coolant temperature and engine RPM

Besides that, we can use LCD to attached to the Arduino board with CAN bus shield and connected to the vehicle OBD-II port and display the values of the parameters of the ECUs on it as we see in figure 67.

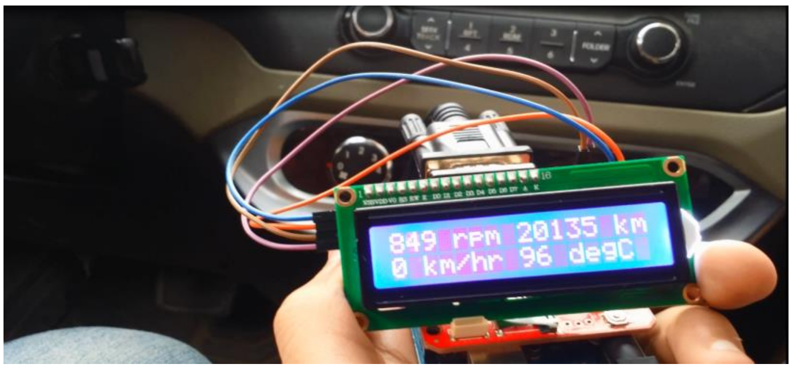


Figure 67 - Displaying the values of engine RPM, distance travelled, vehicle speed, and engine coolant temperature of a stable car

Figure 62 shows 849 RPM on top left that means the engine is running, 0 km/hr speed on bottom left that means the vehicle is immobile, 20135 kilo meters distance travelled on top right and 96° Celsius Engine coolant temperature on bottom right.

By applying more pressure on the fuel throttle, the Engine rotation per minute will increase. As a result of this the vehicle begins to move and its speed is no longer 0 Km/hr as shown in the figure 68.

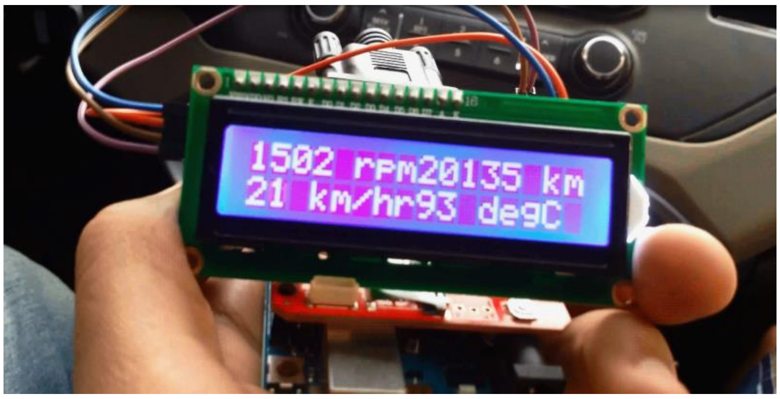


Figure 68 - Displaying the values of engine RPM, distance travelled, vehicle speed, and engine coolant temperature of a moving car

5.3 Reading the DTCs of the Vehicle

As we stated before in chapter 3, DTCs is the codes which is send from the vehicle and decoded in a certain way as we said before in chapter 3, section 3.2.3.

Now we will use the features of mode 3 in the OBD to test if the car has troubles or not.

We will initialize a buffer to add the data in it and a counter to count whether the number of the trouble codes are more than the capacity of the buffer and we have to increase its size or not.

```
char buffer [64];
int count = 0;
```

Then we will send a PID request to the CAN bus using mode 3 and -1 in the PID field because mode 3 does not depend on the PID number.

```
obd.getMultiframePid (0x03, -1, buffer, count);
```

After that we will print on the serial monitor number of DTCs and calculate the counter to enlarge the buffer if that was required.

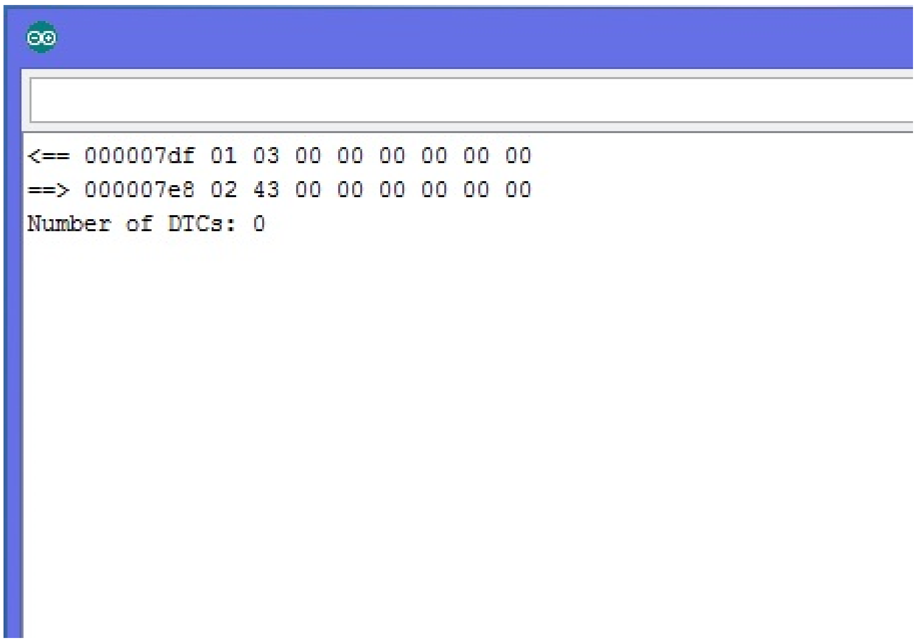
```
Serial.print ("Number of DTCs: ");
Serial.println (count / 2, DEC);
Serial.println ();
```

If the buffer is too small for the number of DTCs, it will be enlarged.

```
if (count > sizeof (buffer)) {
  Serial.println ("Oops, too many. Please increase buffer!");
  for (;;);
}
```

After reading all the DTCs in the buffer, they are ready to be printed on the serial monitor but before printing we have to encode them.

```
for (int i = 0; i < count; i += 2) {
  byte first = (byte) buffer [i];
  byte second = (byte) buffer [i + 1];
  /*The two highest bits encode the subsystem.*/
  switch (first >> 6) {
    case 0: Serial.print ("P"); break;
    case 1: Serial.print ("C"); break;
    case 2: Serial.print ("B"); break;
    case 3: Serial.print ("U"); break;
  }
  /* Next two bits encode the first digit (0-3). */
   Serial.print ((first >> 4) & 0x03, DEC);
    /*Remaining 12 bits are simply three hex digits.*/
   Serial.print (first & 0x0f, HEX);
   Serial.print (second >> 4, HEX);
   Serial.print (second & 0x0f, HEX);
   Serial.println ();
}
```



```
<== 000007df 01 03 00 00 00 00 00 00
==> 000007e8 02 43 00 00 00 00 00 00
Number of DTCs: 0
```

- **5.4 Mobile Application**
- **Introduction**

App Your Car is android mobile application platform. App Your Car provides an easy way where the vehicle's owner can monitor, diagnostic and control his/her car from unlimited distance. App Your Car will notify the vehicle's owner if any error occurs. App Your Car can monitor the speed, RPM …etc. App Your Car is designed for the purpose of achieving the high level of the customer satisfaction.

Figure illustrates the idea of App Your Car application, first the application starts and username and password are required, if they are true the main menu will appear where the vehicle owner can choose between monitoring or controlling or diagnostic his/her car.

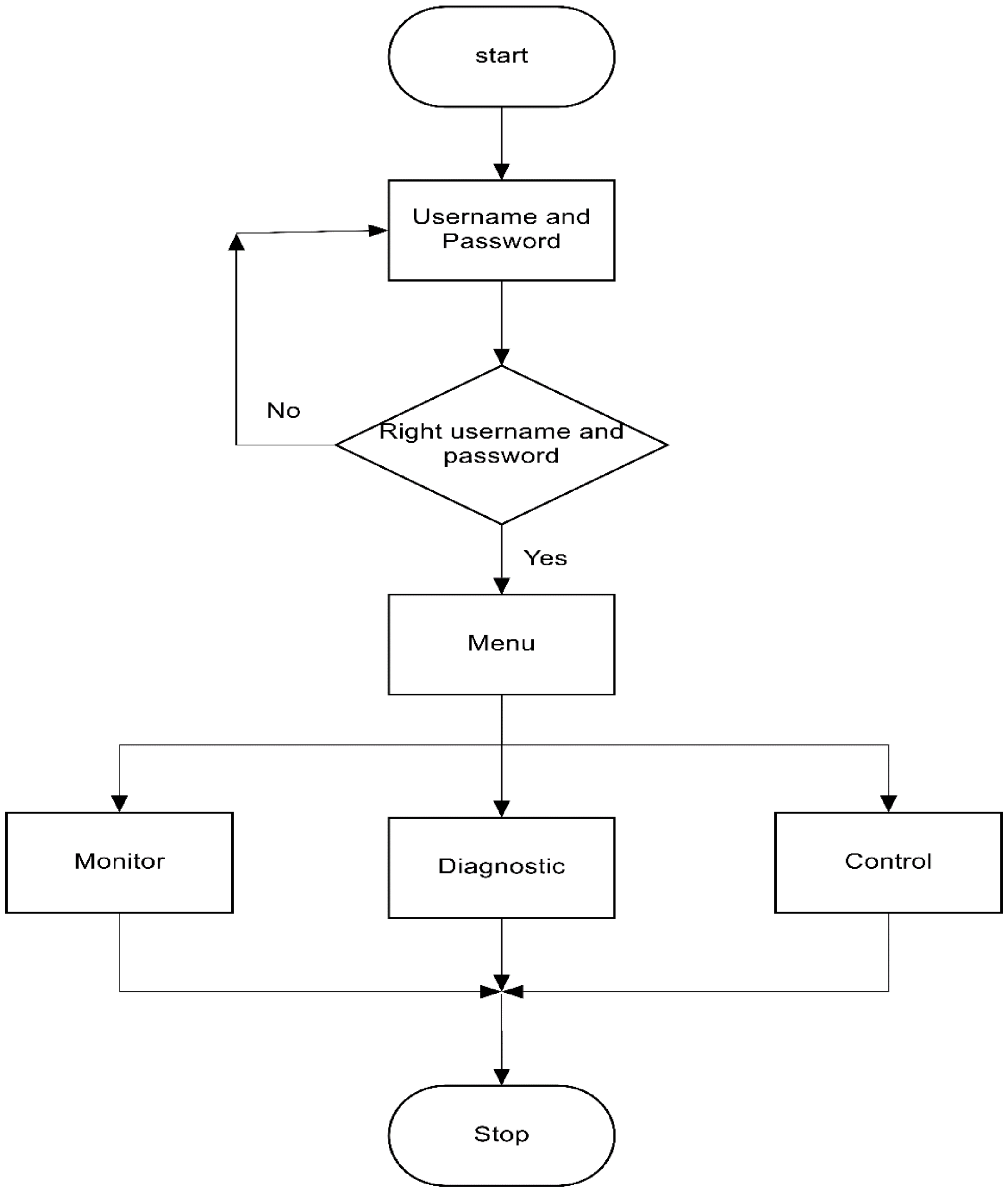


Figure69 - Flow chart of the Android Application

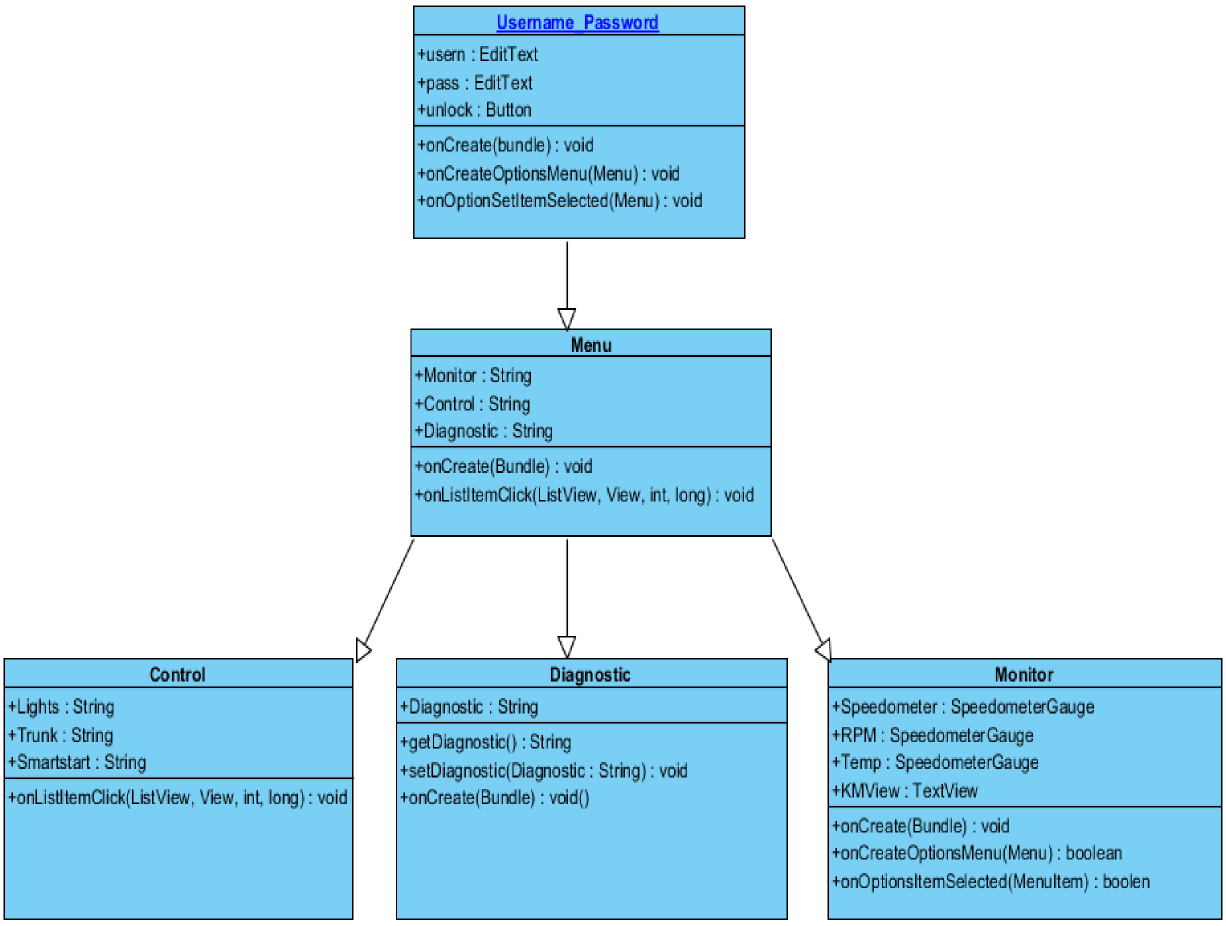

Username_Password
+usern : EditText
+pass : EditText
+unlock : Button
+onCreate(bundle) : void
+onCreateOptionsMenu(Menu) : void
+onOptionSetItemSelected(Menu) : void
Menu
+Monitor : String
+Control : String
+Diagnostic : String
+onCreate(Bundle) : void
+onListItemClick(ListView, View, int, long) : void
Control
+Lights : String
+Trunk : String
+Smartstart : String
+onListItemClick(ListView, View, int, long) : void
Diagnostic
+Diagnostic : String
+getDiagnostic() : String
+setDiagnostic(Diagnostic : String) : void
+onCreate(Bundle) : void()
Monitor
+Speedometer : SpeedometerGauge
+RPM : SpeedometerGauge
+Temp : SpeedometerGauge
+KMView : TextView
+onCreate(Bundle) : void
+onCreateOptionsMenu(Menu) : boolean
+onOptionsItemSelected(MenuItem) : boolen

Figure 70 - The Classes that are used in the application

▪ Classes Used in the Application

- **Username_Password Class**

This class is responsible for checking the user name and password if they are right or not, for more security to the application.

- **Menu Class**

This class is the most important class, in this class the vehicle’s owner can choose between monitoring or controlling or diagnostic.

- **Control Class**

This class is responsible for controlling the ECUs, e.g. opening or closing the lights, open the trunk.

- **Monitoring Class**

This class is responsible for monitoring the reading from the car, e.g. speed of car, RPM, temperature … etc.

- **Diagnostic Class**

This class is responsible for checking your car and knowing any error in the car, this class helps the application to be a mechanic in your pocket.

- **Some Features in App Your Car**

1- App Your Car gives the vehicle’s owner a notification if the engine starts and asks whether the vehicle’s owner wants to stop the engine or not.
2- APP Your Car gives the vehicle’s owner a notification if the speed exceeds 120 Km/hr.
3- APP Your Car gives the vehicle’s owner a notification if the temperature exceeds the normal case.

CHAPTER 6

Implementation & Results

“Duty is ours, results are God's"
Attributed to ***John Quincy Adams***

Finally, after testing all the blocks which we use in our projects individually. And we are sure that each block is working perfectly, it is the time to integrate all those blocks together to get our final output.

By attaching the Arduino Uno board with the CAN bus shield and the Ethernet shield, we paved the road to the data which we read from the vehicle to be displayed on the Android application by using some information about the libraries of the CAN bus shield and Ethernet shield, and how the two shields interface with the Arduino board using <`SPI.h`> library.

The following flow chart in figure 73 illustrates how we organized our work to integrate all tasks together.

▪ 6.1 Problems We Faced and How We Solved it

- **SPI Confliction:**

Arduino Uno board has only one SPI (Serial Peripheral Interface), and the two shields that we are using both of them needs SPI interface. So we have to manage the usage of this serial interface between the two shields respectively without affecting the flow of our output.

The CAN bus shield and the Ethernet shield are using the same chip selection pin on Arduino board (PIN 10). So, we changed the chip selection pin of the CAN bus shield to another pin (SS 9). But we find that is not enough, we have to manually set up the CAN bus CS to be output and to be pulled up to high when it is not used to give a free space to Ethernet board to interface with the Arduino freely.

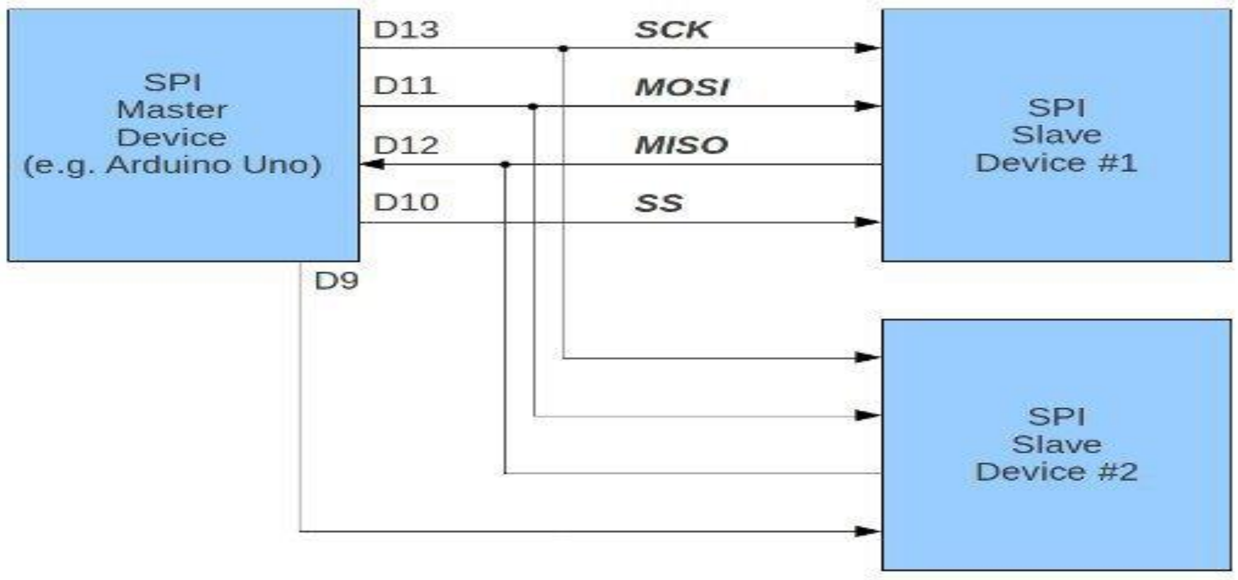


Figure 71 - changing the CS pin from D10 to D9 , Ethernet is Slave Device #1 and CAN shield is Slave Device #2

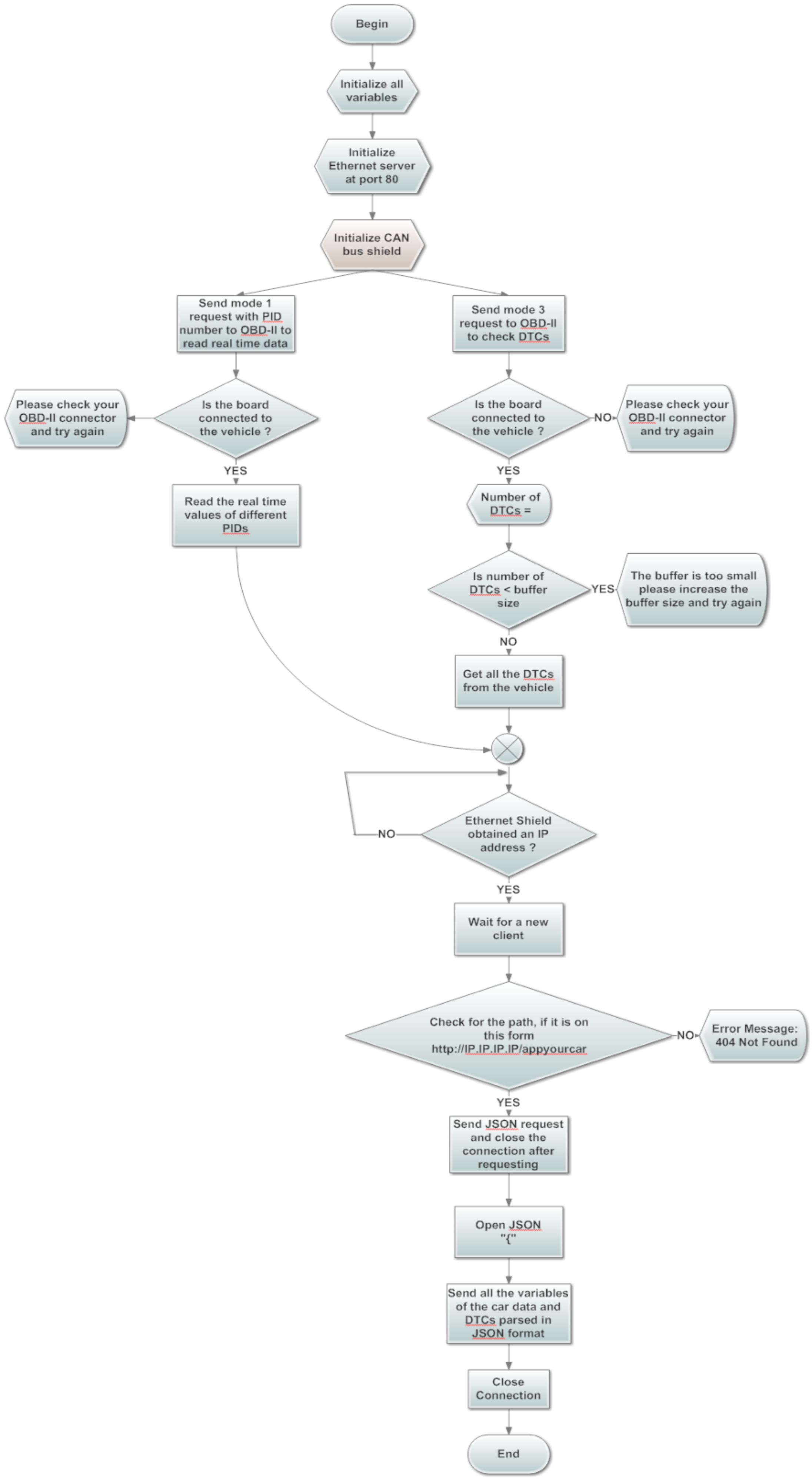

Begin
Initialize all variables
Initialize Ethernet server at port 80
Initialize CAN bus shield
Send mode 1 request with PID number to OBD-II to read real time data
Send mode 3 request to OBD-II to check DTCs
Please check your OBD-II connector and try again
Is the board connected to the vehicle ?
Is the board connected to the vehicle ?
NO
Please check your OBD-II connector and try again
YES
YES
Read the real time values of different PIDs
Number of DTCs =
Is number of DTCs < buffer size
YES
The buffer is too small please increase the buffer size and try again
NO
Get all the DTCs from the vehicle
NO
Ethernet Shield obtained an IP address ?
YES
Wait for a new client
Check for the path, if it is on this form http://IP.IP.IP.IP/appyourcar
NO
Error Message: 404 Not Found
YES
Send JSON request and close the connection after requesting
Open JSON "{"
Send all the variables of the car data and DTCs parsed in JSON format
Close Connection
End

Figure 73 - The flow chart of the whole code

- **Delay Problem**

While reading the values of the PIDs from the ECUs of the vehicle, when we switch from a PID to another one, the PID request take time more than needed to access the ECU and return the reading to the microcontroller. So we have to add a delay after each reading to eliminate this over headed time which gives us a wrong readings or drop down readings from the middle.

- **Memory Problem**

The Arduino Uno board that we used in our project is a microcontroller board based on the ATmega328. The ATmega328 has 32 Kbyte of flash memory for storing code .This amount of memory is not suitable for our code so we used the following ways to minimize our code size:

- Using integer variables instead of float variables.
- Using unsigned character variables instead of character variables.
- Passing flash memory based strings to Serial.print () by wrapping them with F() because strings rapidly consume limited RAM so we warp it to instead store strings in the program flash-memory.

▪ **Adding the 4 Blocks of Codes Together**

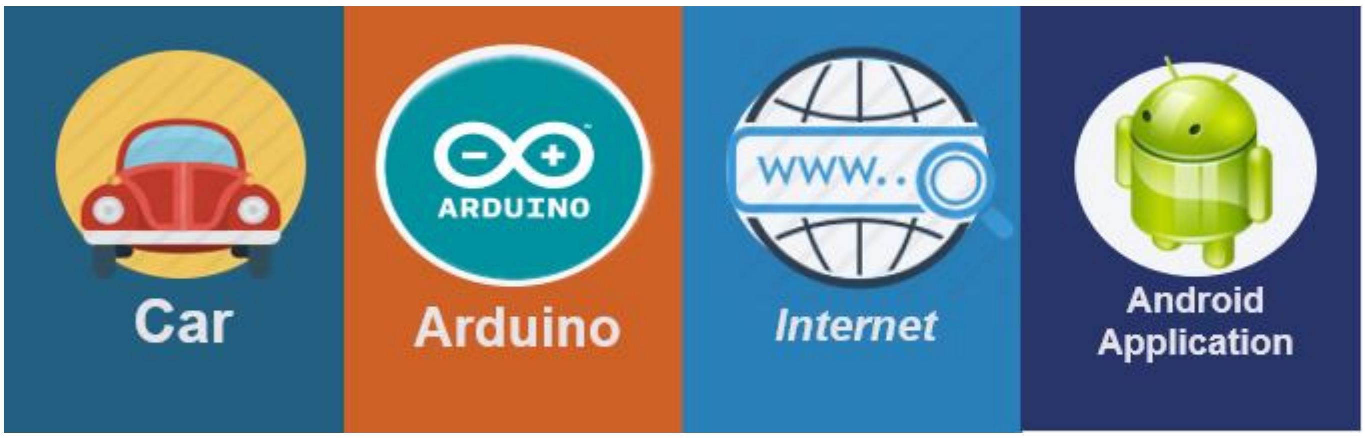


Figure 74 - The main building blocks of the project

As attached in APPENDIX II, we made some modifications on the code of the Ethernet web server. To put it in another way, we replaces the HTML code in the Ethernet Web Server sketch in the Arduino libraries with a JSON code.

Seeing that now the client will be an Android application and not a computer, we have to introduce a new data interchange format rather than HTML, we will use JSON.

- **What is JSON? Why we are going to use it?**

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.
In JSON, they take on these forms:

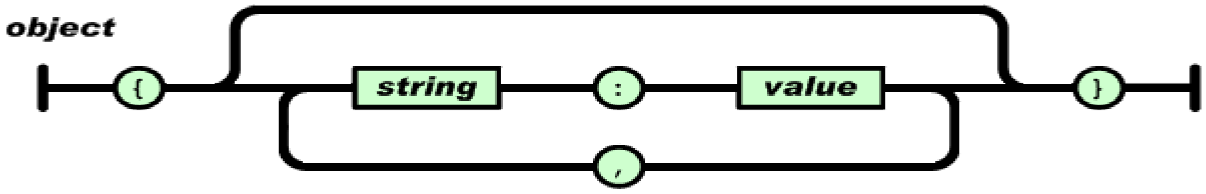


Figure 75 - JSON structre format

An object is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by: (colon) and the name/value pairs are separated by , (comma).

For example, this is a portion of our final code to illustrate how JSON is implemented

```
client.println (F ("HTTP/1.1 200 OK"));
// JSON response type
client.println (F ("Content-Type: application/json"));
// close connection after response
client.println (F ("Connection: close"));
client.println ();
// open JSON
client.print ("{");
// result header
client.print (F ("\"res\":\"OK\""));
// Sending Engine Coolant Temperature value to the Android Application
client.print (F (",\"temp\":\""));
client.print (temp, 1);
client.print (F ("\""));
// Sending Engine RPM value to the Android Application
client.print (F (",\"RPM\":\""));
client.print (rpm, 1);
client.print (F ("\""));
```

At the same time, on the other hand the Android application (client) is waiting for the response of this request. So, we used a library to understand this response and parse it to values which will be displayed on the mobile screen.

- **Android Asynchronous Http Client Library**

An asynchronous callback-based Http client for Android built on top of Apache's HttpClient libraries.
It is used for production by the top applications developers in the world, like Instagram, Pinterest, Pose, etc... [22]

Features of the Library:

- Make asynchronous HTTP requests, handle responses in anonymous callbacks
- `GET/POST` params builder (`RequestParams`)
- Streamed JSON uploads with no additional libraries
- Built-in response parsing into JSON with `JsonHttpResponseHandler`
- Integration with Jackson JSON, Gson or other JSON (de)serializing libraries with `BaseJsonHttpResponseHandler`

- **SpeedometerGauge Component**

Meanwhile, the core of the application is ready to deliver the JSON response the only task left is to display the values in an elegant way.

That is why we used `SpeedometerGauge [23]`component which is a simple needle gauge that looks like speedometer as shown in figure 76.

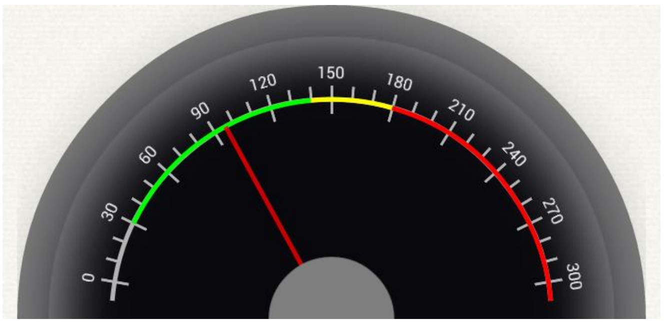


Figure 76 - a simple needle gauge that looks like speedometer

Usage

Import the library to your project.

In your layout xml-file add SpeedometerGauge as shown:

```
<com.cardiomood.android.controls.gauge.SpeedometerGauge
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:padding="8dp"
    android:id="@+id/speedometer" />
```

Then configure the gauge to be similar to the dashboard of the vehicle as shown in figure 77.

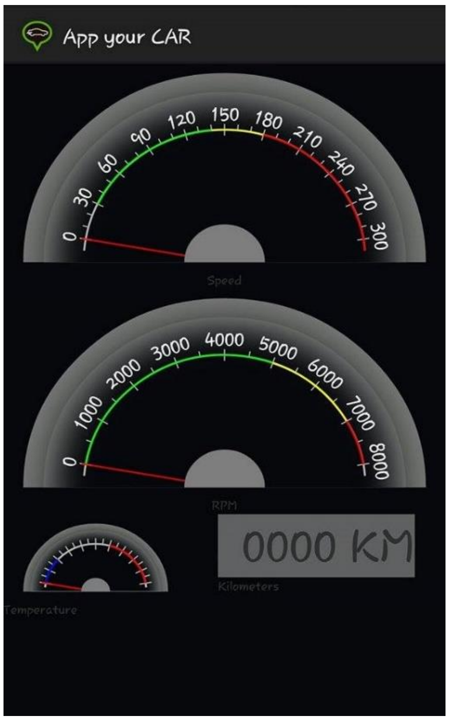


Figure 77 - App Your Car User Interface

CHAPTER 7

Conclusions and Future Work

"One worthwhile task carried to a successful conclusion
is better that 50 half-finished tasks"
Attributed to ***B.C.Forbes***

"The best thing about the future... Is that it comes one day at a time."
Attributed to ***Abraham Lincoln***

▪ Conclusions

We have done our best in this thesis to explain how the automotive industry is playing a vital role in our life. Also to clarify what is the vehicle main components and how they are working together. Through the book progress, we introduced the concept of Automotive Ethernet and showed how we can access vehicle's information through the internet. Additionally we showed that the connection between the mobile and the vehicle has become feasible.

App Your Car can be considered as dashboard and mechanic. The mobile application that is presented by App Your Car is considered as a real time dashboard that monitoring the vehicle readings and displaying them in a smart way on the app through the internet. So no matter where your car is, you will stay connected to it. Besides that the application notifies you by one click if there are any troubles in the car, so it is a mechanic in your pocket.

App Your Car will contribute in automotive industry as it enables the car owner to communicate with his/her vehicle easily via mobile. The automotive companies nowadays are competing to lead the next automotive revolution in vehicles.

Every vehicle will have integrated mobile interface and mobile application to serve it. It also gives the customer the full access to his vehicle by just few clicks on his mobile. Also vehicles will talk to each other's according to the Vehicle-to-vehicle (V2V) concept, which will provide comfortable, prosperity and luxury to our life.

▪ Future Work

We will describe in this chapter the future functionality for our project and how we will develop and make it more efficient and to achieve the customers' satisfaction. This work will not be difficult because our project is designed to be modular in hardware and software design techniques that emphasizes separating the functionality of a program into independent, interchangeable modules.

• Activating the Control Option

In our CAN bus shield, you can add a code to control any parts in the vehicle such as Light, Engine and doors and show them in the mobile application. But, you have to get the Message IDs of these parts to control them .These IDs will be located in the DBC files of your vehicle. The manufacturer of the vehicle only has the rights of these files so you must arrange with them to have the ability of controlling.

- **Add Vehicle Tracking and Position Information**

In our CAN bus shield, there was a socket for adding GPS module that will enable Vehicle tracking and position that will be shown in the mobile application.

- **Support iOS Mobile Operating Systems**

Our mobile application can be implemented as an alternative version to support the iOS operating system for iPhone.

- **Using the SIM Card as a Secure Element in Android**

SIM can be programmed and used as a secure element to enhance the security of Android applications. This approach is very secure because it is difficult to hack the SIM card.

APPENDIX I

▪ PIDs and its Description (Mode 1)

PID (hex)	**Data bytes returned**	**Description**	**Min value**	**Max value**	**Units**
00	4	PIDs supported [01-20]			
01	4	Monitor status since DTCs cleared. (Includes malfunction indicator lamp (MIL) status and number of DTCs.)			
02	2	Freeze DTC			
03	2	Fuel system status			
04	1	Calculated engine load value	0	100	%
05	1	Engine coolant temperature	-40	215	ºC
06	1	Short term fuel % trim—Bank 1	-100 Subtracting Fuel (Rich Condition)	99.22 Adding Fuel (Lean Condition)	%
07	1	Long term fuel % trim—Bank 1	-100 Subtracting Fuel (Rich Condition)	99.22 Adding Fuel (Lean Condition	%
08	1	Short term fuel % trim—Bank 2	-100 Subtracting Fuel (Rich Condition)	99.22 Adding Fuel (Lean Condition)	%
09	1	Long term fuel % trim—Bank 2	-100 Subtracting Fuel (Rich Condition)	99.22 Adding Fuel (Lean Condition)	%
0A	1	Fuel pressure	0	765	kPa (gauge)
0B	1	Intake manifold absolute pressure	0	255	kPa (gauge)
0C	2	Engine RPM	0	16,383.75	rpm
0D	1	Vehicle speed	0	255	km/h
0E	1	Timing advance	-64	63.5	° relative to #1 cylinder
0F	1	Intake air temperature	-40	215	ºC
10	2	MAF air flow rate	0	655.35	grams/sec
11	1	Throttle position	0	100	%
12	1	Commanded secondary air status			
13	1	Oxygen sensors present			

14	**2**	**Bank 1, Sensor 1: Oxygen sensor voltage, Short term fuel trim**	**0 - 100(lean)**	**1.275 99.2(rich)**	**Volts %**
15	2	Bank 1, Sensor 2: Oxygen sensor voltage, Short term fuel trim	0 - 100(lean)	1.275 99.2(rich)	Volts %
16	2	Bank 1, Sensor 3: Oxygen sensor voltage, Short term fuel trim	0 - 100(lean)	1.275 99.2(rich)	Volts %
17	2	Bank 1, Sensor 4: Oxygen sensor voltage, Short term fuel trim	0 - 100(lean)	1.275 99.2(rich)	Volts %
18	2	Bank 2, Sensor 1: Oxygen sensor voltage, Short term fuel trim	0 - 100(lean)	1.275 99.2(rich)	Volts %
19	2	Bank 2, Sensor 2: Oxygen sensor voltage, Short term fuel trim	0 - 100(lean)	1.275 99.2(rich)	Volts %
1A	2	Bank 2, Sensor 3: Oxygen sensor voltage, Short term fuel trim	0 - 100(lean)	1.275 99.2(rich)	Volts %
1B	2	Bank 2, Sensor 4: Oxygen sensor voltage, Short term fuel trim	0 - 100(lean)	1.275 99.2(rich)	Volts %
1C	1	OBD standards this vehicle conforms to			
1D	1	Oxygen sensors present			
1E	1	Auxiliary input status			
1F	2	Run time since engine start	0	65,535	seconds
20	4	PIDs supported [21 - 40]			
21	2	Distance traveled with malfunction indicator lamp (MIL) on	0	65,535	km
22	2	Fuel Rail Pressure (relative to manifold vacuum)	0	5177.265	kPa
23	2	Fuel Rail Pressure (diesel, or gasoline direct inject	0	655,350	kPa (gauge)
24	4	O2S1_WR_lambda(1): Equivalence Ratio Voltage	0 0	1.999 7.999	N/A V
25	4	O2S2_WR_lambda(1): Equivalence Ratio Voltage	0 0	2 8	N/A V
26	4	O2S3_WR_lambda(1): Equivalence Ratio Voltage	0 0	2 8	N/A V
27	4	O2S4_WR_lambda(1): Equivalence Ratio Voltage	0 0	2 8	N/A V
28	4	O2S5_WR_lambda(1): Equivalence Ratio Voltage	0 0	2 8	N/A V
29	4	O2S6_WR_lambda(1): Equivalence Ratio Voltage	0 0	2 8	N/A V

2A	4	O2S7_WR_lambda(1): Equivalence Ratio Voltage	0 0	2 8	N/A V
2B	4	O2S8_WR_lambda(1): Equivalence Ratio Voltage	0 0	2 8	N/A V
2C	1	Commanded EGR	0	100	%
2D	1	EGR Error	-100	99.22	%
2E	1	Commanded evaporative purge	0	100	%
2F	1	Fuel Level Input	0	100	%
30	1	# of warm-ups since codes cleared	0	255	N/A
31	2	Distance traveled since codes cleared	0	65,535	km
32	2	Evap. System Vapor Pressure	-8,192	8,192	Pa
33	1	Barometric pressure	0	255	kPa (Absolute)
34	4	O2S1_WR_lambda(1): Equivalence Ratio Current	0 -128	1.999 127.99	N/A mA
35	4	O2S2_WR_lambda(1): Equivalence Ratio Current	0 -128	2 128	N/A mA
36	4	O2S3_WR_lambda(1): Equivalence Ratio Current	0 -128	2 128	N/A mA
37	4	O2S4_WR_lambda(1): Equivalence Ratio Current	0 -128	2 128	N/A mA
38	4	O2S5_WR_lambda(1): Equivalence Ratio Current	0 -128	2 128	N/A mA
39	4	O2S5_WR_lambda(1): Equivalence Ratio Current	0 -128	2 128	N/A mA
3A	4	O2S7_WR_lambda(1): Equivalence Ratio Current	0 -128	2 128	N/A mA
3B	4	O2S8_WR_lambda(1): Equivalence Ratio Current	0 -128	2 128	N/A mA
3C	2	Catalyst Temperature Bank 1, Sensor 1	-40	6,513.5	ºC
3D	2	Catalyst Temperature Bank 2, Sensor 1	-40	6,513.5	ºC
3E	2	Catalyst Temperature Bank 1, Sensor 2	-40	6,513.5	ºC
3F	2	Catalyst Temperature Bank 2, Sensor 2	-40	6,513.5	ºC
40	4	PIDs supported [41 - 60]			
41	4	Monitor status this drive cycle			
42	4	Control module voltage	0	65.535	V
43	2	Absolute load value	0	25,700	%

44	2	Fuel/Air commanded equivalence ratio	0	2	N/A
45	1	Relative throttle position	0	100	%
46	1	Ambient air temperature	40	215	^{o}C
47	1	Absolute throttle position B	0	100	%
48	1	Absolute throttle position C	0	100	%
49	1	Accelerator pedal position D	0	100	%
4A	1	Accelerator pedal position E	0	100	%
4B	1	Accelerator pedal position F	0	100	%
4C	1	Commanded throttle actuator	0	100	%
4D	2	Time run with MIL on	0	65,535	minutes
4E	2	Time since trouble codes cleared	0	65,535	minutes
4F	4	Maximum value for equivalence ratio, oxygen sensor voltage, oxygen sensor current, and intake manifold absolute pressure	0, 0, 0, 0	255, 255, 255, 2550	V, mA, kPa
50	4	Maximum value for air flow rate from mass air flow sensor	0	2550	g/s
51	1	Fuel Type			
52	1	Ethanol fuel %	0	100	%
53	2	Absolute Evap system Vapor Pressure	0	327.675	kPa
54	2	Evap system vapor pressure	0	32,768	Pa
55	2	Short term secondary oxygen sensor trim bank 1 and bank 3	-100	99.22	%
56	2	Long term secondary oxygen sensor trim bank 1 and bank 3	-100	99.22	%
57	2	Short term secondary oxygen sensor trim bank 2 and bank 4	-100	99.22	%
58	2	Long term secondary oxygen sensor trim bank 2 and bank 4	-100	99.22	%
59	2	Fuel rail pressure (absolute)	0	655,350	KPa
5A	1	Relative accelerator pedal position	0	100	%
5B	1	Hybrid battery pack remaining life	0	100	%
5C	1	Engine oil temperature	-40	210	^{O}C
5D	2	Fuel injection timing	-210.00	301.992	-
5E	2	Engine fuel rate	0	3212.75	L/h
5F	1	Emission requirements to which vehicle is designed			
60	4	PIDs supported [61 - 80]			
61	1	Driver's demand engine - percent torque	-125	125	%
62	1	Actual engine - percent torque	-125	125	%
63	2	Engine reference torque	0	65,535	Nm
64	5	Engine percent torque data	-125	125	%
65	2	Auxiliary input / output supported			
66	5	Mass air flow sensor			
67	3	Engine coolant temperature			
68	7	Intake air temperature sensor			
69	7	Commanded EGR and EGR Error			
6A	5	Commanded Diesel intake air flow control and relative intake air flow position			
6B	5	Exhaust gas recirculation temperature			
6C	5	Commanded throttle actuator control and relative throttle position			

6D	6	Fuel pressure control system			
6E	5	Injection pressure control system			
6F	3	Turbocharger compressor inlet pressure			
70	9	Boost pressure control			
71	5	Variable Geometry turbo (VGT) control			
72	5	Waste gate control			
73	5	Exhaust pressure			
74	5	Turbocharger RPM			
75	7	Turbocharger temperature			
76	7	Turbocharger temperature			
77	5	Charge air cooler temperature (CACT)			
78	9	Exhaust Gas temperature (EGT) Bank 1			
79	9	Exhaust Gas temperature (EGT) Bank 2			
7A	7	Diesel particulate filter (DPF)			
7B	7	Diesel particulate filter (DPF)			
7C	9	Diesel Particulate filter (DPF) temperature			
7D	1	NOx NTE control area status			
7E	1	PM NTE control area status			
7F	13	Engine run time			
80	4	PIDs supported [81 - A0]			
81	21	Engine run time for Auxiliary Emissions Control Device(AECD)			
82	21	Engine run time for Auxiliary Emissions Control Device(AECD)			
83	5	NOx sensor			
84		Manifold surface temperature			
85		NOx reagent system			
86		Particulate matter (PM) sensor			
87		Intake manifold absolute pressure			
A0	4	PIDs supported [A1 - C0]			
C0	4	PIDs supported [C1 - E0]			
C3	?	?	?	?	?
C4	?	?	?	?	?

APPENDIX II

- **Supported Parameters**

```
/**********************************************************************
* AppYourCar - Your Mechanic in Your Pocket
* Copyright (C) 2015 AppYourCar team,
* Cairo University, Faculty of Engineering
* Supported Parameters Sketch
* Circuit:
* Sparkfun CAN bus shield + OBD-II to DB9 Cable
* Arduino Uno
**********************************************************************/
#include <AppYourCar.h>
ObdInterface obd;
ObdMessage msg;
void setup() {
  Serial.begin(115200);
  obd.setSlow (false);
  obd.setExtended (false);
  obd.setDebug (false);
  obd.begin ();
  Serial.println ();
  Serial.println ();
  Serial.println ("   | 0 1 2 3 4 5 6 7 8 9 A B C D E F");
  Serial.println ("---+------------------------------------");
  for (int i = 0; i <= 0xf0; i += 0x10) {
    Serial.print (i, HEX);
    Serial.print (i == 0 ? "0 |" : " |");
    for (int j = 0; j <= 0x0f; j++) {
      boolean supported;
      if (obd.isPidSupported (i + j, supported)) {
        Serial.print (supported ? " X" : "  ");
      } else {
        Serial.print (" ?");
      }
      delay (50);
    }
    Serial.println ();
  }
}
void loop () {
}
```

- **Trouble Codes**

```
/***********************************************************************
* AppYourCar - Your Mechanic in Your Pocket
* Copyright (C) 2015 AppYourCar team,
* Cairo University, Faculty of Engineering
* Trouble Codes Sketch
* Circuit:
* Sparkfun CAN bus shield + OBD-II to DB9 Cable
* Arduino Uno
***********************************************************************/
#include <AppYourCar.h>

ObdInterface obd;

void setup () {
  Serial.begin (115200);
  obd.setSlow (false);
  obd.setExtended (false);
  obd.setDebug (true);
  obd.begin ();
}

void loop () {
  char buffer [64];
  int count = 0;

  if (obd.getMultiframePid(0x03, -1, buffer, count)) {

    Serial.print ("Number of DTCs: ");
    Serial.println (count / 2, DEC);
    Serial.println ();

    if (count > sizeof (buffer)) {

      Serial.println ("Oops, too many. Please increase buffer!");
      for (;;);

    }

    /*Iterate through the buffer and decode all DTCs
     *
     * according to what Wikipedia tells us:
     * http://en.wikipedia.org/wiki/OBD-II_PIDs#Bitwise_encoded_PIDs
     */

    for (int i = 0; i < count; i += 2) {
      byte first = (byte) buffer[i];
      byte second = (byte) buffer[i + 1];
      /* The two highest bits encode the subsystem. */

      switch (first >> 6) {
        case 0: Serial.print ("P"); break;
        case 1: Serial.print ("C"); break;
        case 2: Serial.print ("B"); break;
        case 3: Serial.print ("U"); break;
      }
      /* Next two bits encode the first digit (0-3).*/

      Serial.print ((first >> 4) & 0x03, DEC);

      /* Remaining 12 bits are simply three hex digits. */

      Serial.print (first & 0x0f, HEX);
      Serial.print (second >> 4, HEX);
      Serial.print (second & 0x0f, HEX);
      Serial.println ();

    }
  }
  for (;;);
}
```

- **Full Code**

```
#include <Ethernet.h>
#include <AppYourCar.h>
#include <SPI.h>

// debug ON-OFF
#define UARTDEBUG 1

ObdInterface obd;
char _buf[50];
int ethok = 0;
byte mac[] =  { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip;

boolean isreqline = true;
String req = String();
String par = String();

// Ethernet server
EthernetServer server(80);

// the setup routine runs once when you press reset:
void setup() {
  // open serial communications and wait for port to open:
  Serial.begin(250000);
  pinMode (9, OUTPUT);
  digitalWrite(9, HIGH);
  Serial.println("ECU Reader");  /* For debug use */
  obd.setSlow(false);
  obd.setExtended(false);
  obd.setDebug(false);
  obd.begin();
  while (ethok == 0) restartEthernetShield();
}


void restartEthernetShield() {
  delay(100);
  if (UARTDEBUG) Serial.println("-- RESTART ETHERNET SHIELD");
  ethok = Ethernet.begin(mac);
  if (ethok == 0) {
    if (UARTDEBUG) Serial.println ("-- ERROR : failed to configure
Ethernet");
  } else {
    // extract ip
    ip = Ethernet.localIP();
    // start server
    server.begin();
    // start listening for clients
    if (UARTDEBUG) {
      formatIP(ip);
      Serial.print("-- IP ADDRESS is ");
      Serial.println(_buf);
    }
  }
  delay(100);
}


void loop() {

  unsigned int temp, carSpeed, throttle, load, distance, runTime, rpm;

  char buffer[64];
  int count = 0;

  if (obd.getMultiframePid(0x03, -1, buffer, count)) {
    Serial.print("Number of DTCs: ");
    Serial.println(count / 2, DEC);
    Serial.println();
    int conter = count/2 ;

    if (count > sizeof (buffer)) {
      Serial.println("Oops, too many. Please increase buffer!");
      for (;;);
    }
    /*
     * Iterate through the buffer and decode all DTCs
     * according to what Wikipedia tells us:
     *
     * http://en.wikipedia.org/wiki/OBD-II_PIDs#Bitwise_encoded_PIDs
     */

    for (int i = 0; i < count; i += 2) {
      byte first = (byte)buffer[i];
      byte second = (byte)buffer[i + 1];

      /*
       * The two highest bits encode the subsystem.
       */
      switch (first >> 6) {
        case 0: Serial.print("P"); break;
        case 1: Serial.print("C"); break;
        case 2: Serial.print("B"); break;
        case 3: Serial.print("U"); break;
      }

      /*
       * Next two bits encode the first digit (0-3).
       */
      Serial.print((first >> 4) & 0x03, DEC);

      /*
       * Remaining 12 bits are simply three hex digits.
       */
      Serial.print(first & 0x0f, HEX);
      Serial.print(second >> 4, HEX);
      Serial.print(second & 0x0f, HEX);

      Serial.println();
    }
  }



////////////////////////////////////////////////////////////////////////
//////////////////// GETTING READINGS FROM THE VEHICLE/////////////////
////////////////////////////////////////////////////////////////////////

  obd.getPidAsInteger(0x05, temp);          // Engine Coolant Temprature
  obd.getPidAsInteger(0x0c, rpm);           // Engine RPM
  obd.getPidAsInteger(0x0d, carSpeed);      // Vehicle Speed
  obd.getPidAsInteger(0x11, throttle);      // Throttle Position
  obd.getPidAsInteger(0x04, load);          // Engine Load
  obd.getPidAsInteger(0x21, distance);      // Distance Travelled
  obd.getPidAsInteger(0x1F, runTime);      // Run time since engine start
```

```
  Serial.print (F("Engine Coolant Temperature =  "));
  Serial.println (temp);
  Serial.print (F("Engine RPM = "));
  Serial.println (rpm);
  Serial.print (F("Vehicle Speed = "));
  Serial.println (carSpeed);
  Serial.print (F("Throttle Position = "));
  Serial.println (throttle);
  Serial.print (F("Engine Load = "));
  Serial.println (load);
  Serial.print (F("Distance Travelled = "));
  Serial.println (distance);
  Serial.print (F("Run Time = "));
  Serial.println (runTime);

  if (ip[0] != 0) {
    EthernetClient client = server.available();   // listen to client
connecting
    if (client) {
      // new client http request
      boolean currentLineIsBlank = true;   // an http request ends with
a blank line
      while (client.connected()) {
        if (client.available()) {
          // read char from client
          char c = client.read();
          // append to request string
          if ((isreqline) && (req.length() < 127)) req += c;
          // stop parsing after first line
          if (c == '\n') isreqline = false;

          // if you've gotten to the end of the line (received a newline
character) and the line is blank,
          // the http request has ended, so you can send a reply
          if ((c == '\n') && currentLineIsBlank) {

            // if request does not contain "appyourcar" keyword send 404
            if (req.indexOf("appyourcar") == -1) send_404(client);
            // else send JSON response
            else {

              // response header
              client.println( F("HTTP/1.1 200 OK"));
              // JSON response type
              client.println (F("Content-Type: application/json"));
              // close connection after response
              client.println (F("Connection: close"));
              client.println ();
              // open JSON
              client.print("{");
              // result header
              client.print (F("\"res\":\"OK\""));
              formatMAC();
              client.print (F(",\"hwid\":\""));
              client.print (_buf);
              client.print (F("\""));
              // temperature
              client.print (F(",\"temp\":\""));
              client.print (temp, 1);
              client.print (F("\""));
              // RPM
              client.print (F(",\"RPM\":\""));
              client.print (rpm, 1);
              client.print (F("\""));
              // Speed
              client.print (F(",\"Speed\":\""));
              client.print (carSpeed, 1);
              client.print (F("\""));
              // Distance Travelled
              client.print (F(",\"KM\":\""));
              client.print (distance, 1);
              client.print (F("\""));
               // load
              client.print (F(",\"Load\":\""));
              client.print (load, 1);
              client.print (F("\""));
              // Runtime
              client.print (F(",\"Runtime\":\""));
              client.print (runTime/60, 1);
              client.print (F("\""));
              // Throttle
              client.print (F(",\"Throttle\":\""));
              client.print (throttle, 1);
              client.print (F("\""));
              // DTC
              client.print (F(",\"DTC\":\""));
              client.print (counter, 1);
              client.print (F("\""));
              // pressure
              client.print (F(",\"pres\":\""));
              client.print (0);
              client.print (F("\""));
              // close json
              client.println(F("}"));
            }

            // prepare for next request
            req = "";
            isreqline = true;
            // exit
            break;
          }
          if (c == '\n') {
            // you're starting a new line
            currentLineIsBlank = true;
          }
          else if (c != '\r') {
            // you've gotten a character on the current line
            currentLineIsBlank = false;
          }
        }
      }
      // give the web browser time to receive the data
      for (int xx = 0; xx <= 16000; xx++)
      { }
      // close the connection:
      client.stop();
    }
  }
}

void formatIP(IPAddress ii) {
  sprintf(_buf, "%d.%d.%d.%d", ii[0], ii[1], ii[2], ii[3]);
}

void formatMAC() {
  sprintf(_buf, "%02x:%02x:%02x:%02x:%02x:%02x", mac[0], mac[1], mac[2],
mac[3], mac[4], mac[5]);
}

static void send_404(EthernetClient client) {
  client.println(F("HTTP/1.1 404 Not Found"));
  client.println(F("Content-Type: text/html"));
  client.println();
  client.println(F("404 NOT FOUND"));
```

APPENDIX III

- **Budget**

Component	Quantity	Source	Price
Spark fun CAN-BUS Shield	1	SparkFun Electronics USA	310 LE
OBD-II to DB9 Cable	1	SparkFun Electronics USA	40 LE
Ethernet Shield	1	Future Electronics Egypt	120 LE
Arduino Uno R3	1	Future Electronics Egypt	150 LE
Arduino USB programming Cable	1	Future Electronics Egypt	5 LE
TP-LINK 150MBPS PORTABLE 3G ROUTER WITH BATTERY	1	Computer Shop Egypt	333 LE
Vodafone USB Modem	1	Vodafone Egypt	125 LE
Total			**1083 LE**

References

[1] F. Bonomi, "The Smart and Connected Vehicle and the IoT," 2013. [Online]. Available: http://tf.nist.gov/seminars/WSTS/PDFs/1-0_Cisco_FBonomi_ConnectedVehicles.pdf.

[2] C. M. Kozierok, C. Correa, R. B. Boatright and J. Quesnelle, Automotive Ethernet: The Definitive Guide, USA: Intrepid Control Systems., 2014.

[3] M. Di Natale, Understanding and using the Controller Area, 2008.

[4] P. Bagschik, "An Introduction to CAN," I-ME ACTIA, Germany, 2008.

[5] "CAN history," CAN-CIA, 1 July 2015. [Online]. Available: http://www.can-cia.de/index.php?id=161.

[6] P. Richards, "A CAN Physical Layer Discussion," Microchip Technology Inc, USA, 2002.

[7] Renesas Electronics, "Introduction to CAN," Renesas Electronics, 2012.

[8] Autotap, "OBD-II Background," Autotap, 2011. [Online]. Available: http://www.obdii.com/background.html.

[9] Wikipedia, "On-board diagnostics," Wikipedia, the free encyclopedia, [Online]. Available: https://en.wikipedia.org/wiki/On-board_diagnostics.

[10] Wikipedia, the free encyclopedia, "OBD-II PIDs," Wikipedia, the free encyclopedia, [Online]. Available: https://en.wikipedia.org/wiki/OBD-II_PIDs.

[11] Team ZR1, "Project Description," [Online]. Available: http://teamzr1.com/project/mode.html.

[12] Alex, "Complete List of OBD Codes: Generic OBD2 (OBDII) & Manufacturer," Total Car Diagnostics support, 29 Jan 2013. [Online]. Available: http://www.totalcardiagnostics.com/support/Knowledgebase/Article/View/21/0/genericmanufacturer-obd2-codes-and-their-meanings.

[13] Real Time Logic, "What is an Embedded Application Server?," Real Time Logic, 12 September 2012. [Online]. Available: https://realtimelogic.com/blog/2012/09/What-is-an-Embedded-Application-Server.

[14] W. Nilsen, "Get on the Internet of Things fast with an embedded Web app server," Embedded, Cracking the code to systems development, 1 January 2014. [Online]. Available: http://www.embedded.com/design/programming-languages-and-tools/4426418/Get-on-the-Internet-of-Things-fast-with-an-embedded-Web-app-server--Part-1.

[15] ATMEL, "Embedded Web Server," ATMEL Corporation 2002.

[16] S. Lazim, Design an Embedded Web Server for Road Traffic Monitoring, Mosul: College of Electronics Engineering, University of Mosul, 2013.

[17] J. Axelsom, Embedded Ethernet and Internet Complete, Lakeview Research LLC, 2003.

[18] Wikipedia, "Transmission Control Protocol (TCP)," Wikipedia, The free Enclopedia, [Online]. Available: https://en.wikipedia.org/wiki/Transmission_Control_Protocol.

[19] Microchip, "TCP vs. UDP," Microchip, [Online]. Available: https://microchip.wikidot.com/tcpip:tcp-vs-udp.

[20] Wikipedia, "User Datagram Protocol (UDP)," Wikipedia, Tthe free encyclopedia, [Online]. Available: https://en.wikipedia.org/wiki/User_Datagram_Protocol.

[21] G. Johns, R. Hurtado, B. Harris and Z. Pham, Auto Performance Analyzer, USA, Texas: Department of Computer Science and Engineering, The University of Texas at Arlington, 2013.

[22] J. Smith, "Android Asynchronous Http Client," [Online]. Available: http://loopj.com/android-async-http/.

[23] ntoskrnl, "AndroidWidgets," 2014. [Online]. Available: https://github.com/ntoskrnl/AndroidWidgets.