Palestine Polytechnic University

College of Engineering & Technology
Electrical and Computer Engineering Department

Graduation Project

Using Mobile Phone As a Car Remote Control

Project Team
Ayman Samamreh
Abdullah Awawdeh
Ihab Salameen

Project Supervisor
Eng. Amal Al-Dweik Wazwaz

Hebron – Palestine

June, 2008

جامعة بوليتكنك فلسطين

الخليل – فلسطين

كلية الهندسة و التكنولوجيا

دائرة الهندسة الكهربائية و الحاسوب

أسم المشروع

Using Mobile Phone As a Car Remote Control

أسماء الطلبة

ايمن سمامره عبدالله عواوده ايهاب السلامين

بناء على الأنظمة المتبعة في كلية الهندسة و التكنولوجيا و إشراف و متابعة المشرف المباشر على المشروع و موافقة اعضاء اللجنة الممتحنة تم تقديم هذا المشروع إلى دائرة الهندسة الكهربائية و الحاسوب وذلك للوفاء بمتطلبات درجة البكالوريس في الهندسة تخصص هندسة أنظمة الحاسوب

توقيع المشرف

.....

توقيع اللجنة الممتحنة

....

توقيع رئيس الدائرة

.....

Abstract

This Project is about using the mobile phone as an advanced Remote Control for a car, by installing a central locking circuit that uses Bluetooth for communication, a user with Bluetooth enabled mobile phone and the appropriate software could lock/unlock the doors and the switch, and to turn the engine on/off.

The project uses Bluetooth technology to send commands to the car, as a Bluetooth already exists in the mobile phone, one Bluetooth module is needed at the car, the Bluetooth connection has ranges of 1, 30, or 100 meters, and the implemented range in this project is 30 meters.

The project design has been implemented in hardware and software. The final testing was done on a real car and most results were as expected; the range of the system was about 20 meters without obstacles between the car and the phone, and down to 10 meters with obstacles.

Dedication

To

Our Parents

Acknowledgment

This work could not have been accomplished without the help of many people; the team would like to thank:

Palestine Polytechnic University and College of Engineering & Technology;

The staff of the Department of Electrical & Computer Engineering;

Our supervisor Engineer Amal Al-Dweik Wazwaz;

Our friends and families.

Table of Contents

List of Figures

List of Tables

Chapter One

Introduction

1.1 General Idea about the Project

1.2 Project Objectives

1.3 System and User Requirements

1.4 Literature Review

1.5 Estimated Cost

1.6 Risk Management

1.7 Time Plan

1.8 Report Contents

1.1 General Idea about the Project

This project is about making use of mobile phone as an advanced remote control for the car, it will control the central locking system and the cars key switch. Also, the mobile phone will be used to turn on/off the engine using Bluetooth as transmitting medium.

An important aspect of the project is to use the mobile phone as a general purpose remote control. Also, it provides a good replacement for the cars ordinary remote control (as ordinary remote control units have a limited functionality) and provides the ability of turning on/off the car engine.

Another important issue is that making use of Bluetooth in a car opens the way for many new ideas in the automation area. Some of these ideas are provided in the future work.

1.2 Project Objectives

This projects aims at achieving the following objectives:

- Use mobile phone as a remote control.
- Develop an advanced remote control for the car.
- Interface mobile phone with control system using Bluetooth technology.

1.3 System and User Requirements

1.3.1 System Requirements

When a user chooses an operation (door lock) the program should send a corresponding code to the control unit, the control unit interprets the code and carry out the operation meant by the code.

1.3.2 User Requirements

- Remotely lock/unlock car doors.
- Remotely lock/unlock car switch.
- Turn car engine on/off.

1.3.3 Non-Functional Requirements

The system must achieve the following:

- Reliability: A mobile phone for a specific car should not work with another car that has the system installed.
- Security: Only the owner of the phone should be able to use it with his car, not anyone else who could access the phone.
- Flexibility: The system should work with any Nokia S60 2nd edition or later phone, not only with a specific model, and should be more flexible than ordinary remote control.
- Simple user interface.

1.4 Literature Review

The most application implemented by Bluetooth is wireless data transfer, especially voice transfer. Another less common application is the transfer of the remote control signals; follows are some works that implements Bluetooth remote control, using mobile phone or specially built circuit:

Blue-Control: A Bluetooth Remote Control for the Sony Ericsson T610 Mobile Phone

The aim of this project was to create an expandable framework for utilizing Bluetooth enabled mobile devices to control applications running under MS Windows. [1]

In this project two software programs were developed, one is running on a MS Windows, and the other on a mobile phone. No hardware was built; a Bluetooth dongle attached to the computer, and a T610 mobile phone.

Bluetooth Remote Control... From Your Mobile Phone

Using mobile phone as a remote unit that logs temperature, locks door with a password, and a tracking device, wirelessly via Bluetooth. [2]

This project contains three parts; in first part a Sony Ericson smart phone is used to read and log temperature via Bluetooth from a built circuit, in the second part any smart phone or Bluetooth enabled computer is used to lock/unlock a garage door with a password, in the last part a handheld computer is used to control a simple robot.

The hardware of this project is not built from scratch, instead a ready development board that contains a Bluetooth module and a microcontroller is used.

Bluetooth Remote Control for the Traffic Applications

A Bluetooth handheld device used as an expedient tool for traffic police on the road to control Urban Traffic Control System. [3]

In this project a handheld device was built so that traffic police could use it to monitor and change the mode of operation such as Auto, Manual, and Flashing without opening Police Control Panel. This project is not educational; it's developed by National Research Development Corporation in India.

Blue-wand: A Versatile Remote Control and Pointing Device

The Blue-wand is a small, pen-like device that can be used to control Bluetooth enabled devices by hand-movements. [4]

In this project a handheld device is built, this device uses Bluetooth to send the hand movement (hand moved left, right, up, down, forward, or backward) to a Bluetooth

enabled computer or smart phone. The device could be used as a remote control to for many consumer electronics (TV, MP3…), access control for machines and buildings, or for video gaming. No receiving device is implemented, just the handheld device (the transmitter).

1.5 Estimated Cost

The total estimated budget for the systems is about 220 USD, divided as follows:

Component	Required Number	Price (U.S Dollar)
Parani ESD200 Bluetooth	1	80
Smart Phone	1	120
PIC18F4550 Microcontroller	1	15
Central Looking System	1	30
Basic Electronic Components	--	*20*

Table 1.1 Estimated Costs

1.6 Risk Management

1.6.1 Project Risks

Major risks we have determined for this project are as follows:

- Illness/Death of a Staff Member.
- Computer Crash
- Late Delivery
- Technology will not Meet Specifications
- Changes in Requirements
- Lack of Development Experience

1.6.2 Risk Mitigation, Monitoring and Management

Risk: Computer Crash

- **Mitigation**

  The cost associated with a computer crash resulting in a loss of data is crucial. A computer crash itself is not crucial, but rather the loss of data. A loss of data will result in not being able to deliver the product. As a result we are taking steps to make multiple backup copies of our works in development and all documentation associated with it, in multiple locations.

- **Monitoring**

  When working on the project or documentation, we should always be aware of the stability of the computing environment they are working in. Any changes in the stability of the environment should be recognized and taken seriously.

- **Management**

  The lack of a stable-computing environment is extremely hazardous to a software development team. In the event that the computing environment is found unstable, we should cease work on that system until the environment is made stable again, or should move to a system that is stable and continue working there.

Risk: Illness/Death of a Staff Member

- **Mitigation**

  The problem associated with an illness/death of a staff member is crucial.
  An illness/death of a staff member will result in not being able to deliver our project in the time.

- **Monitoring**

  When working on the project or documentation, the staff member will divide the works on three instead of two; this will reduce the effect of having shortage in our staff.

- **Management**

  The illness/death of a staff member is extremely hazardous to our team. Having external staff is not allowable.

Risk: Late Delivery

- **Mitigation**

  The cost associated with a late delivery is critical. A late delivery will result in receiving a failing grade for the course. Steps have been taken to ensure a timely delivery by estimating the scope of project based on the delivery deadline.

- **Monitoring**

  A schedule has been established to monitor project status. Falling behind schedule would indicate a potential for late delivery. The schedule will be followed closely during all development stages.

- **Management**

  Late delivery would be a catastrophic failure in the project development. If the project cannot be delivered on time then we will not pass the course. If it becomes apparent that the project will not be completed on time, the only course of action available would be to request an extension to the deadline from the university.

Risk: **Technology Does Not Meet Specifications**

- **Mitigation**

  In order to prevent this from happening, meetings will be held with our supervisor. This insures that the project we are developing and the specifications of the supervisor are equivalent.

- **Monitoring**

  The meetings with the supervisor should ensure that the supervisor and our team understand each other and the requirements for the project.

- **Management**

  Should the development team come to the realization that their idea of the product specifications differs from those of the supervisor and university, the supervisor should be immediately notified and whatever steps necessary to rectify this problem should be done. Preferably a meeting should be held between our team and the supervisor to discuss at length this issue.

Risk: **Changes in Requirements**

- **Mitigation**

  In order to prevent this from happening, meetings will be held with our supervisor.

  This insures that the project we are developing and the specifications of the supervisor are equivalent.

- **Monitoring**

  The meetings with the supervisor should ensure that the supervisor and the university and our team understand each other and the requirements for the project.

- Management

  Should the development team come to the realization that their idea of the product specifications differs from those of the supervisor and university, the supervisor should be immediately notified and whatever steps necessary to rectify this problem should be done. Preferably a meeting should be held between our team and the supervisor to discuss at length this issue.

Risk: **Lack of Development Experience**

- Mitigation

  In order to prevent this from happening, our team will be required to learn the languages and techniques necessary to develop this project. The member of our team that is the most experienced in a particular facet of the development tools will need to instruct those who are not as well versed.

- Monitoring

  Each member of our team should watch and see areas where another team member may be weak. Also if one of the members is weak in a particular area it should be brought to the attention by that member, to the other members.

- Management

  The members who have the most experience in a particular area will be required to help those who don't. In some circumstances, the experienced member will take the rule of the inexperienced member.

1.7 Time Plan

Task / week	3	4	5	6	7	8	9	10	11	12	13	14	15
Collecting Information about the Project													
Studying Bluetooth Technology													
Studying KC21 & PIC Microcontroller													
Studying Car Central Locking System													
Studying Mobile													
Documentation Writing													

Table 1.2 First Semester Time Plan

Task / week	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Building the Central Locking Circuit														
Interfacing Bluetooth														
Programming the Smart Phone														
System Testing														
Documentation Writing														

Table 1.3 Second Semester Time Plan

1.8 Report Contents

Chapter 1: Introduction.

The first chapter provides a general idea about the project and lists the project objectives, next some previous works are provided, and then the time schedule and

the estimated budget breakdown are given. Finally the project risks are listed there management plan.

Chapter 2: Theoretical Background.

The second chapter is a theoretical background related to the main idea of the project and its software and hardware components.

Chapter 3: Project Conceptual Design.

This chapter describes project objectives in some detail, the overall system design options as well as its components are provided, the general block diagram is described, and the how the system interacts with its surrounding environment.

Chapter 4: Detailed Technical Project Design.

This chapter provides a detailed description of the main three phases (input, processing and output) for the project as a whole. Then a description of the subsystem detailed design and schematics. The complete system schematic is given next with a brief description of the system design. And lastly a view of the system user interface.

Chapter 5: Software System Design.

This chapter gives a description of the design of the microcontroller program, the program which will receive and decode the message send from mobile phone, send command to the Bluetooth module, and also will initiates commands to the car's center locking system hardware. Also the design of the smart phone program is discussed, and the software packages used.

Chapter 6: System Implementation and Testing.

Chapter six discusses the actual implementation of the project, and the various testing stages of the system; components testing, subsystem testing and integrated testing.

Chapter 7: Software System Design.

In chapter seven, the conclusion is given along with some suggestion and development for future work.

Chapter Two

Theoretical Background

2.1 Introduction

2.2 Theoretical Background

2.3 Project Integrity

2.4 Theoretical Background about Project Components

2.5 Summary

2.1 Introduction

The main idea of this project is to use the mobile phone as an advanced remote control for a car, a program installed on the phone will use the phone's Bluetooth to communicate with another Bluetooth module installed in the car, the car's Bluetooth will pass received commands to a microcontroller that will interact with car's hardware.

In this chapter we will discuss the main technologies that will be used in designing and implementing the system

2.2 Theoretical Background

2.2.1 The Mobile Phone

A Smartphone is a mobile phone offering advanced capabilities beyond a typical mobile phone, often with PC-like functionality. There is no industry standard definition of a Smartphone. For some, a Smartphone is a phone that runs complete operating system software providing a standardized interface and platform for application developers. For others, a Smartphone is simply a phone with advanced features. [5]

According to the above definition, The Mobile Phone used in the project could also be called Smartphone as it runs a complete OS and has some advanced features.

2.2.1.1 Mobile Phone Software

2.2.1.1.1 *Symbian OS*

Symbian OS is the advanced, open operating system licensed by the world's leading mobile phone manufacturers. It is designed for the specific requirements of advanced 2G, 2.5G and 3G (a fully digital cellular telecoms networks that are based on the Global System for Mobile Communications) mobile phones. Symbian OS combines the power of an integrated applications environment with mobile telephony, bringing advanced data services to the mass market. [6]

Symbian OS Architecture

Figure 2.1 Symbian OS Architecture

Kernel

The kernel is the central manager and arbiter of Symbian OS. It manages the system memory and schedules programs for execution. It also allocates shared system resources and handles any functionality that requires privileged access to the CPU. The kernel can be extended via Dynamic Link Libraries (DLLs) and device drivers.

Base libraries

The base libraries contain APIs that provide functionality such as string manipulation, linked lists, file I/O, database management, error handling and timers. The base libraries also provide access to kernel functions (e.g. thread control and client server communications). This library is used not only by applications, but also by the OS components.

Application framework

The application framework implements the base functionality of the phone's graphical user interface applications. This includes a framework for handling the GUI itself and an architecture framework for handling non-GUI related application functionality.

Communications architecture

The communications architecture consists of the APIs and framework that implement data communications. This includes TCP/IP over cellular radio as well as local communication protocols such as Bluetooth, IR, and USB. Also included is the messaging framework for support that includes SMS, MMS and email messaging. [7]

Developing on Symbian OS

There are multiple platforms, based upon Symbian OS that provide an SDK for application developers wishing to target a Symbian OS device – the main ones being UIQ and S60. Individual phone products, or families, often have SDKs or SDK extensions downloadable from the manufacturer's website too. The SDKs contain documentation, the header files and library files required to build Symbian OS software, and a Windows-based emulator ("WINS"). Up until Symbian OS version 8, the SDKs also included a version of the GCC compiler (a cross-compiler) required to build software to work on the device.

Symbian OS 9 uses a new ABI[1] and so requires a new compiler – a choice of compilers is available including a new version of GCC. In terms of SDKs, UIQ Technology now provides a simplified framework so that the single UIQ SDK forms the basis for developing on all UIQ 3 devices, such as the Sony Ericsson P990 and Sony Ericsson M600.

Symbian C++ programming is commonly done with an IDE. For previous versions of Symbian OS, the commercial IDE CodeWarrior for Symbian OS was favored. The CodeWarrior tools were replaced during 2006 by Carbide.c++, an Eclipse-based IDE developed by Nokia. Microsoft Visual Studio 2003 and 2005 also supported through Carbide.vs plug-in.

Visual Basic, VB.NET, and C# development for Symbian were possible through AppForge Crossfire, a plug-in for Microsoft Visual Studio.

There is also a version of a Borland IDE for Symbian OS. Symbian OS development is also possible on Linux and Mac OS X using tools and techniques developed by the community, partly

[1] Application Binary Interface

enabled by Symbian releasing the source code for key tools. A plug-in that allows development of Symbian OS applications in Apple's Xcode IDE for Mac OS X is available.

Once developed, Symbian OS applications need to find a route to customers' mobile phones. They are packaged in SIS files which may be installed over-the-air, via PC connect or in some cases via Bluetooth or memory cards. An alternative is to partner with a phone manufacturer to have the software included on the phone itself. The SIS file route is more difficult for Symbian OS 9.x, because any application wishing to have any capabilities beyond the bare minimum must be signed via the Symbian Signed program.

Java ME applications for Symbian OS are developed using standard techniques and tools such as the Sun Java Wireless Toolkit (formerly the J2ME Wireless Toolkit). They are packaged as JAR (and possibly JAD) files. Both CLDC and CDC applications can be created with NetBeans. Other tools include SuperWaba, which can be used to build Symbian 7.0 and 7.0s programs using Java. Nokia S60 phones can also run python scripts when the interpreter is installed, with a custom made API that allows for Bluetooth support and such. There is also an interactive console to allow the user to write python scripts directly from the phone. [8]

2.2.2 S60 Platform

The S60 Platform (formerly Series 60 User Interface) is a software platform for mobile phones that uses Symbian OS. S60 is currently amongst the leading Smartphone platforms in the world. It is developed primarily by Nokia and licensed by them to other manufacturers.

There are a few common features in S60:

- Devices' display resolution is originally 176x208. Since 2nd Edition Feature Pack 3, S60 supports multiple resolutions, i.e. Basic (176x208), QVGA (240x320) and Double (352x416).
- It supports Java (J2ME MIDP 2.0 commonly, but varies from phone to phone.) applications and Symbian C++ applications.
- Certain buttons are standardized, such as left and right select, Menu, Clear, and Input Settings.

There have been three releases of S60: "Series 60" (2001), "Series 60 Second Edition" (2004) and "Series 60 3rd Edition" (2005).

It is noteworthy that software written for S60 1st edition (S60v1) or 2nd edition (S60v2) is not binary compatible with S60 3rd edition (S60v3), because it uses a new, hardened version of the Symbian OS (v9.1). [9]

2.2.1.3 Symbian C++

C++ is the primary language for software development on Symbian OS since it provides the most efficient and natural interface to the system level frameworks and APIs which themselves are written in C++. In fact, Symbian OS itself is written almost entirely in C++. When developing Symbian software, you'll be using many of the standard C++ language features, including inheritance, encapsulation, virtual functions, function overloading, and templates. These language features are not only used for implementing your application logic, but also in using the system APIs. For example, some APIs are abstract classes that your application classes can inherit from and extend their functionality as needed. Other APIs are classes that are instantiated and used directly. Still others are simple function calls implemented as static class methods that can be called directly in the same manner as C-based APIs – no class instantiation is required (the static API class User is a good example of this).

Although Symbian OS uses many of the object-oriented features of C++, some of its functionality is implemented in nonstandard ways. This can require an adjustment, even for experienced C++ programmers. For example, Symbian implements its own exception-based mechanism for handling errors such as low memory conditions in place of the C++ throw/catch exception feature. Also, Symbian OS does not use the Standard Template Library (STL) and instead has Symbian OS-specific implementation classes for functions such as string manipulation and complex collection types. Symbian decided on this course for a variety of reasons including making the implementation more efficient for resource constrained devices. [7]

2.2.2 Wireless Networking

While the term wireless network may technically be used to refer to any type of network that is wireless, the term is most commonly used to refer to a telecommunications network whose interconnections between nodes is implemented without the use of wires, such as a computer network (which is a type of communications network). Wireless telecommunications networks are generally implemented with some type of remote information transmission system that uses electromagnetic waves, such as radio waves, for the carrier and this implementation usually takes place at the physical level or layer of the network. [10]

There are many types of wireless networks; most common are wireless LAN, wireless MAN, and wireless PAN.

A Personal Area Network (PAN) is a computer network used for communication among computer devices (including telephones and personal digital assistants) close to one person. The devices may or may not belong to the person in question. The reach of a PAN is typically a few meters. PANs can be used for communication among the personal devices themselves (intrapersonal communication), or for connecting to a higher level network and the Internet (an uplink).

The standards dealing with wireless LAN, MAN, and PAN are referred to IEEE 802. The services and protocols specified in IEEE 802 map to the lower two layers (Data Link and Physical) of the seven-layer OSI networking reference model.

IEEE 802 is divided into a number of working groups, each working on a specific technology related to networking, the 11^{th} group (802.11), for example, deals with WLAN, and the 15^{th} group (802.15) deals with WPAN.

2.2.2.1 Bluetooth

Bluetooth wireless technology is a short-range communications system intended to replace the cable(s) connecting portable and/or fixed electronic devices. The key features of Bluetooth

wireless technology are robustness, low power, and low cost. Many features of the core specification are optional, allowing product differentiation.

The Bluetooth core system consists of an RF transceiver, baseband, and protocol stack. The system offers services that enable the connection of devices and the exchange of a variety of classes of data between these devices. [11]

Bluetooth media access is based on a frequency-hopping scheme allowing many independent point-to-multipoint connections in the same physical space. Through separate inquiry and paging processes, a master device searches for and connects with a slave device in range, resulting in a piconet.

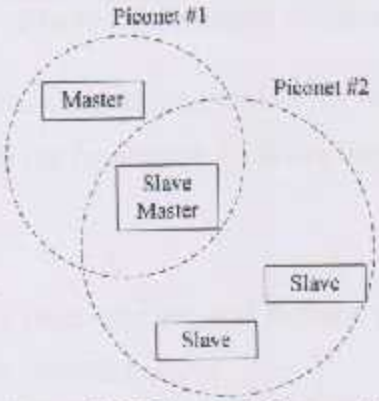

Figure 2.2 Bluetooth Connection

An interesting case arises when a master device must allow for new incoming connections. Such a device can advertise itself as a potential slave, and when a connection occurs, it may request a master-slave switch, ensuring its role as master for all connections. Figure 2.4 illustrates the temporary scatternet that occurs in this case.

2.2.2.1.1 Bluetooth Profiles

In order to use Bluetooth wireless technology, a device must be able to interpret certain Bluetooth profiles. The profiles define the possible applications. Bluetooth profiles are general behaviors through which Bluetooth enabled devices communicate with other devices. Bluetooth technology defines a wide range of profiles that describe many different types of use cases. By

following a guidance provided in Bluetooth specifications, developers can create applications to work with other devices also conforming to the Bluetooth specification.

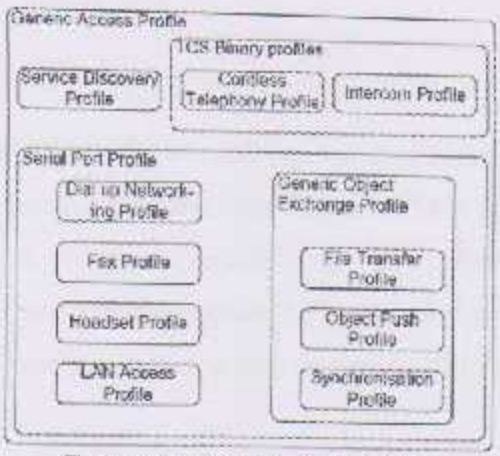

Figure 2.3 Bluetooth Profiles

The two general Bluetooth profiles described next form the basis for any user models and their profiles. The profiles also provide the foundation for future user models and profiles.

Generic Access Profile (GAP)

GAP provides the basis for all other profiles and defines a consistent means to establish a baseband link between Bluetooth enabled devices.

In addition to this, GAP defines the following:

- The features must be implemented in all Bluetooth devices
- Generic procedures for discovering and linking to devices
- Basic user-interface terminology

GAP ensures a high degree of interoperability between applications and devices. It also makes it easier for developers to define new profiles by leveraging existing definitions.

GAP handles discovery and establishment between devices that are unconnected. The profile defines operations that are generic and can be used by profiles referring to GAP and by devices implementing multiple profiles. GAP ensures that any two Bluetooth enabled devices, regardless of manufacturer and application, can exchange information via Bluetooth technology in order to

discover what type of applications the devices support. Bluetooth enabled devices not conforming to any other Bluetooth profile must conform to GAP to ensure basic interoperability and co-existence.

Serial Port Profile (SPP)

SPP defines how to set-up virtual serial ports and connect two Bluetooth enabled devices. SPP is based on the ETSI TS07.10 specification and uses the RFCOMM protocol to provide serial-port emulation. SPP provides a wireless replacement for existing RS-232 based serial communications applications and control signals. SPP provides the basis for the DUN, FAX, HSP and LAN profiles. This profile supports a data rate up to 128 Kbit/sec. SPP is dependent on GAP.

2.2.2.1.2 *Bluetooth Protocol Stack*

The heart of the Bluetooth specification is the Bluetooth protocol stack. By providing well-defined layers of functionality, the Bluetooth specification ensures interoperability of Bluetooth devices and encourages adoption of Bluetooth technology. As seen in Figure 2.3, these layers range from the low-level radio link to the profiles.

At the base of the Bluetooth protocol stack is the radio layer. The radio module in a Bluetooth device is responsible for the modulation and demodulation of data into RF signals for transmission in the air. The radio layer describes the physical characteristics a Bluetooth device's receiver-transmitter component must have. These include modulation characteristics, radio frequency tolerance, and sensitivity level.

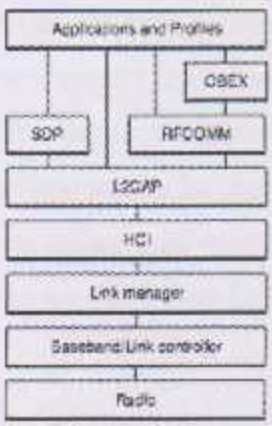

Figure 2.4 Bluetooth Protocol Stack

Bluetooth Core Protocols

Baseband

The Baseband and Link Control layer enables the physical RF link between Bluetooth units forming a piconet. This layer controls the Bluetooth unit's synchronization and transmission frequency hopping sequence. The two different link types defined in Bluetooth, Synchronous Connection Oriented, SCO, and Asynchronous Connectionless, ACL, described in the section Link types are also managed by this layer. The ACL links, for data, and the SCO links, mainly for audio, can be multiplexed to use the same RF link.

Host Controller Interface, HCI

The Host Controller Interface, HCI, provides a uniform interface method for accessing the Bluetooth hardware capabilities. It contains a command interface to the Baseband controller and link manager and access to hardware status. Finally, it contains control and event registers.

Link Manager Protocol, LMP

The Link Manager Protocol, LMP, is responsible for link set-up between Bluetooth units. It handles the control and negotiation of packet sizes used when transmitting data. The Link Manager Protocol also handles management of power modes, power consumption, and state of a Bluetooth unit in a piconet. Finally, this layer handles generation, exchange and control of link and encryption keys for authentication and encryption.

Service Discovery Protocol, SDP

The Service Discovery Protocol, SDP, defines how a Bluetooth client's application shall act to discover available Bluetooth servers' services and their characteristics. The protocol defines how a client can search for a service based on specific attributes without the client knowing anything of the available services. The SDP provides means for the discovery of new services becoming available when the client enters an area where a Bluetooth server is operating. The SDP also provides functionality for detecting when a service is no longer available.

Cable Replacement Protocols

RFCOMM

The RFCOMM protocol is a serial port emulation protocol. The protocol covers applications that make use of the serial ports of the unit. RFCOMM emulates RS-232 control and data signals over the Bluetooth baseband. It provides transport capabilities for upper level services, e.g. OBEX that use a serial line as the transport mechanism.

Adopted protocols

Many other protocols are defined to be adapted, this includes but not limited to: PPP (Point-to-Point Protocol), TCP/UDP/IP, OBEX (Object Exchange), WAP (Wireless Access Protocols). [12]

2.2.2.1.3 Bluetooth Air Interface

To meet the requirements for the air interface a frequency band between 2.400 and 2.500 GHz was selected. Thus, the requirements regarding operating worldwide support for both data and speech and the limitations regarding physical characteristics (size and power consumption) were covered. This radio frequency band is the Industrial-Scientific-Medical, ISM band and ranges in Europe and the USA from 2.400 to 2.4835 GHz (in France and Spain only parts of this range are available). As a result, Bluetooth devices must be able to act in the range from 2.400 to 2.500 GHz and be able to select a segment in the ISM band within which they can act. The ISM band is open to any radio system. Cordless telephones, garage door openers and microwave ovens operate in this band, where microwave ovens are the strongest source of interference. Bluetooth units connect to each other forming a so-called piconet, consisting of up to eight active Bluetooth units.

Frequency hopping is literally jumping from frequency to frequency within the ISM band. After a Bluetooth device sends or receives a packet, it and the Bluetooth device or devices it is

communicating with "hop" to another frequency before the next packet is sent. This scheme has three advantages:

- It allows Bluetooth devices to use the entirety of the available ISM band, while never transmitting from a fixed frequency for more than a very short time this ensures that Bluetooth conforms to the ISM restrictions on transmission quantity per frequency.
- It ensures that any interference will be short-lived. Any packet that doesn't arrive safely at its destination can be sent again at the next frequency.
- It provides a base level of security because it's very difficult for an eavesdropping device to predict which frequency the Bluetooth devices will use next.

The connected devices must agree upon the next frequency to use. The Bluetooth specification ensures this in two ways. First, it defines a master-slave relationship between Bluetooth devices. Second, it specifies an algorithm that uses device-specific information to calculate frequency-hop sequences.

A Bluetooth device operating in master mode can communicate with up to seven slave devices. To each of its slaves, the master Bluetooth device sends its own unique device address (similar to an Ethernet address) and the value of its internal clock. This information is used to calculate the frequency-hop sequence. Because the master device and all its slaves use the same algorithm with the same initial input, the connected devices always arrive together at the next frequency.

Power Consumption

As a cable-replacement technology, it's not surprising that Bluetooth devices are usually battery-powered devices, such as wireless mice and mobile phones. To conserve power, most Bluetooth devices operate as low-power, 1 mW radios (Class 3 radio power). This gives Bluetooth devices a range of about 5–10 meters. This range is far enough for comfortable wireless peripheral communication but close enough to avoid drawing too much power from the device's power source. With 2.5 mW (Class 2 radio power) the range is extended to 10–30 meters. And with 100 mW (Class 1 radio power) the range is about 100 meters.

Bluetooth Networking

When Bluetooth units are communicating, one unit is master and the rest of the units act as slaves. The master unit's system clock and the master identity are the central parts in the frequency hop technology. The hop channel is determined by the hop sequence and by the phase in this sequence. The identity of the master determines the sequence and the master unit's system clock determines the phase. In the slave unit, an offset may be added to its system clock to create a copy of the master's clock. In this way every unit in the Bluetooth connection holds synchronized clocks and the master identity, that uniquely identifies the connection. Hop carriers have been defined for the Bluetooth technology except for France and Spain where 23 hop carriers have been defined, because the ISM-band is narrower there.

The Bluetooth packets have a fixed format. A 72-bit access code comes first in the packet. The access code is based on the master's identity and the master's system clock, i.e. it provides the means for the synchronization. This code is unique for the channel and used by all packets transmitting on a specific channel. A 54-bit header follows the access code. This header contains error correction, retransmission and flow control information. The error correction information can be used for correcting faults in the payload and in the header itself. Finally comes the payload field with anything between zero and 2,745 bits, i.e. up to 340 bytes.

2.2.2.1.4 Bluetooth Advantages

The Bluetooth concept offers several benefits compared with other techniques. The main advantages of Bluetooth are:

- The minimal hardware dimensions
- The low price on Bluetooth components
- The low power consumption for Bluetooth connections

2.2.3 Central Locking System

The central locking system locks and unlocks the doors as well as the luggage and engine compartments. When the doors are locked, inside release knobs are all lowered. If airbags inflate in an accident, all doors are automatically unlocked. Additional safety settings offer extra convenience: automatic locking of the luggage compartment or safety central locking. Then all doors and the luggage compartment are then locked automatically when the car is driven at a speed of 10 kilometers an hour or faster for example. The luggage compartment locks when the speed is greater than 5 km/h and the safety central locking only initially allows only the driver's door to be opened. A second opening movement opens the remaining doors as well as the tailgate.

The central locking system consists of a collection of wires and motors, and a control unit, the control unit receives messages from a remote control, interprets the signal, and sends electrical currents to the motors connected to it. The communication between the remote control and the control unit is done via Radio Frequency.

Certain part of this project is to design a control unit that uses Bluetooth to connect to the remote control (a mobile phone in this case)

Figure 2.5 Central Locking System

2.2.3.1 Door Locking Circuit

When the door is turned in the driver's door-lock, all the other doors on the vehicle should also lock. Motors usually achieve this.

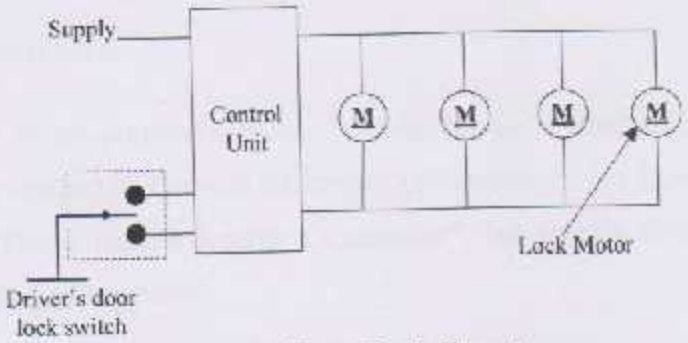

Figure 2.6 Door Lock Circuit

Figure 2.8 shows door lock circuit; a main control unit contains relays which are actuated by a door lock switch or the remote infrared key (remote control), the motors at each door are simply wired in parallel and all operate at the same time.

Infrared central door locking is controlled by a small hand-held transmitter and an infrared receiver unit as well as a decoder in the main control unit. This layout will vary slightly between different manufactures. When the infrared key is operated by pressing a small switch, a complex code is transmitted. The infrared sensor picks up this code and sends it in an electrical form to the main control unit. If the received code is correct, the relays are triggered and the door-locks are either locked or unlocked. If an incorrect code is received on three consecutive occasions when attempting to unlock the door, then the infrared system will switch itself off until the door is opened by key. This will also reset the system and allow the correct code to operate the lock again. This technique prevents a scanning type transmitter of being used to open the doors. [13]

2.3 Project Integrity

The wireless communication in the system is carried out by a Bluetooth class 2 device (maximum power output is 2.5mW), in relation to health concern, class 2 devices are considered of less potential hazard than mobile phone for example.

The system also will increase the car's safety by locking the switch with a password system that is only accessible by the user's phone.

2.4 Theoretical Background about Project Components

2.4.1 PIC Microcontroller

Harvard architecture microcontrollers made by Microchip Technology, derived from the PIC1640 originally developed by General Instrument's Microelectronics Division. The name PIC initially referred to "Programmable Interface Controller", but shortly thereafter was renamed "Programmable Intelligent Computer".

PICs are popular with developers and hobbyists alike due to their low cost, wide availability, large user base, extensive collection of application notes, availability of low cost or free development tools, and serial programming (and re-programming with flash memory) capability.

2.5 Summary

In this chapter a theoretical background related to the main idea of the project including a detailed discussion of smart phones technology and an overview of the wireless communication and networking and Bluetooth technology was presented followed by a theoretical background about project components.

Chapter Three

Conceptual Design

3.1 Introduction

3.2 Detailed Project Objectives

3.3 Design Options

3.4 Design Realization Approach

3.5 Project Design Block Diagram

3.6 System Modeling

3.7 Project Interaction with the Surrounding Environment

3.8 Summary

3.1 Introduction

In this chapter a detailed description of the project objectives and the design block diagram is given. Also the various design options for the system and its components are discussed. The data flow diagram and the use case sequence diagram are provided. Finally a description about how the system interacts with the surrounding environment.

3.2 Detailed Project Objectives

- Use the mobile phone as a remote control.
  Write software that will make use of the phones Bluetooth to control another device.
- Develop an advanced remote for the car.
  Most of the cars have limited remote controls that lock/unlock doors; the advanced remote control will extend this function so that it will lock/unlock the switch and start or shutdown the car engine.
- Interface the mobile phone with a control unit using Bluetooth technology.
  Use the Bluetooth as a connection medium between the phone and car.

3.3 Design Options

3.3.1 System Design Options

Below is the possible ways of interfacing the mobile phone with the control unit installed in the car:

3.3.1.1 Using Infrared

In this option a mobile phone with infrared capability is needed to connect to another infrared unit installed in the car, the main disadvantage here is that no obstacle should exist between the

car and the phone or the infrared connection will not be available. Also, few mobile phones have built in infrared port.

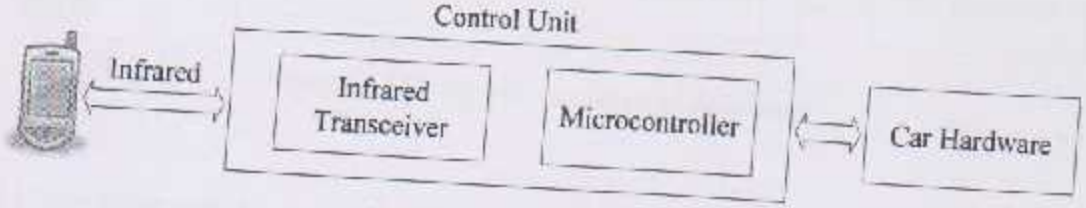

Figure 3.1 Using Infrared

3.3.1.2 Using Peer Mobile through GSM Network

The remote mobile phone communicates with a peer phone installed in the car by sending messages (SMS) via Global System for Mobile Communications (GSM) network. The advantage of such design is that the car can be controlled from any place in the world. A disadvantage is that controlling the car won't be free of charge, because of sending SMS.

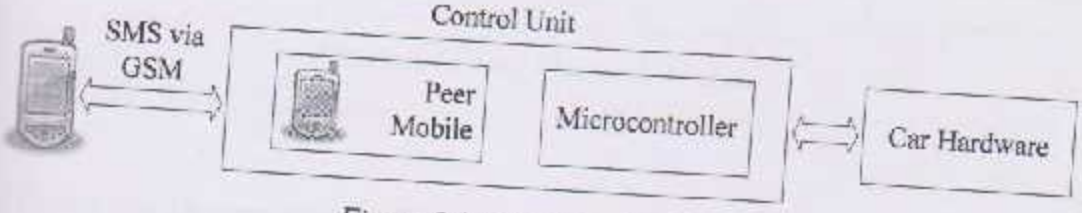

Figure 3.2 Using Peer Mobile

3.3.1.3 Using Bluetooth Microcontroller

A mobile phone with Bluetooth technology will connect to a microcontroller which has built in Bluetooth. This will overcome the direct connection problem faced with infrared, but, although a microcontroller with Bluetooth will simplify the control circuit, the price of such microcontroller is expensive.

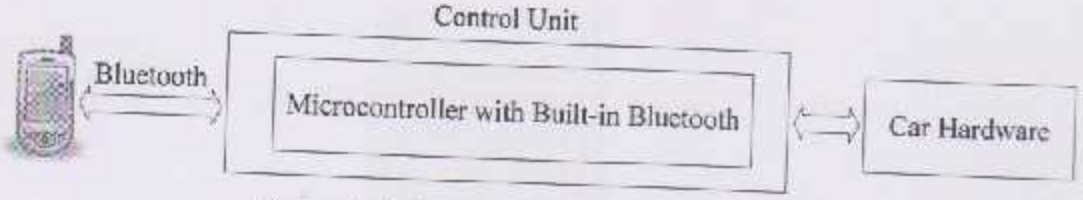

Figure 3.3 Using Microcontroller Bluetooth

3.3.1.4 Using Bluetooth Module

A mobile phone with Bluetooth technology will connect to a Bluetooth module which is connected to a microcontroller. The Bluetooth module is responsible for communicating with the mobile phone, while the microcontroller will interact with the car's hardware. This the chosen design because most mobile phones have Bluetooth, and Bluetooth modules are cheap.

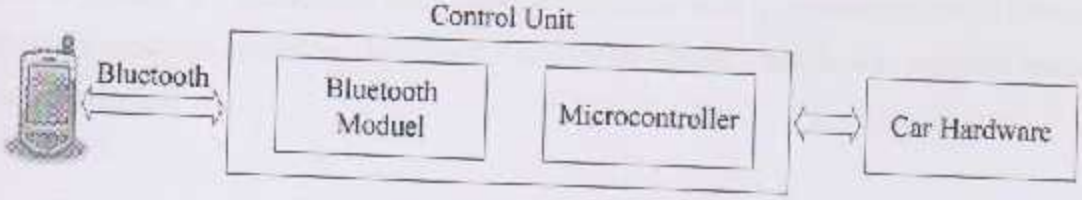

Figure 3.4 Using Bluetooth Module

3.3.2 Hardware Design Options

3.3.2.1 Mobile Phone

The options for a mobile phone are limited, the Nokia 6630 smart phone was chosen because most of smart phones in the local market are form Nokia, and the 6630 model is already available with the team. The Nokia 6630 is Nokia's first Third Generation phone based on Symbian OS 8.1. It features always-connected email, mobile broadband access for multimedia content, live video streaming, video conferencing, and Bluetooth 1.2. It has 220 MHz processor, 20 MB built-in memory, and 512 MB MMC memory card.

Figure 3.5 Nokia 6630 Smart Phone

3.2.2 Bluetooth Module

The Bluetooth module is a transceiver that connects the phone with a microcontroller. There are many Bluetooth modules available; the Parani ESD200 Bluetooth module was selected among the following modules:

Ericson Bluetooth module ROK 101 107

Figure 3.6 ROK 101 107

The Ericson Bluetooth module was used in most of previous projects related to Bluetooth, one good advantage of it that it can communicate with the host device through USB, UART and RS-232. We cannot find any place that sells this, as the manufacturing of it seems to be discontinued.

Sena Parani-ESD200 Bluetooth module

Figure 3.7 Parani-ESD200

The Parani-ESD200 is Class 2 type of Compact Embedded Bluetooth Serial Modules for RS-232 cable replacement. It supports 30 meters of wireless transmit distance. This module was chosen because it's available in the market.

KC-Wirefree KC-21 Bluetooth module

Figure 3.8 KC-21

One of the most capable Bluetooth modules available, the KC-21 Bluetooth OEM (Original Equipment Manufacturer) Module is designed for maximum flexibility. The KC-21 module includes 14 general purpose input/output lines, and offers high speed serial communications up to 921K baud. The KC-21 is a surface mount PCB module that provides fully embedded, ready to use Bluetooth wireless technology. The reprogrammable flash memory contains embedded firmware for serial cable replacement using the Bluetooth SPP profile. Other popular Bluetooth profiles are available. [14]

The KC-21 has a built-in 48MHz ARM7 microprocessor with 8MB flash memory, built-in antenna, and range up to 20 meter.

3.2.3 Microcontroller

The choices of a microcontroller are directly related to the chosen Bluetooth module, as the ESD200 was chosen, a microcontroller that is capable of communicating with it is needed. The

ESD200 is able to communicate with the host device using the UART, so any microcontroller with a serial port is good.

The PIC18F4550 was chosen because of its following features:

- Ideal for low power (nanoWatt) and connectivity applications.
- The availability of three serial ports: FS-USB(12Mbit/s), I²C (Inter-Integrated Circuit) and SPI (Serial Port Interface) up to 10 Mbit/s, and an asynchronous serial port EUSART (Enhanced Universal Synchronous/Asynchronous Receiver Transmitter).
- Large amounts of RAM memory for buffering and Enhanced Flash program memory make it ideal for embedded control and monitoring applications that require periodic connection with a (legacy free) personal computer via USB for data upload/download and/or firmware updates. [15]

3.3.3 Software Design Options

3.3.3.1 Operating System

The chosen operating system is directly related to the smart phone. Two popular operating systems were considered, the Microsoft Windows Mobile, and the Symbian OS.

Microsoft Windows Mobile is much easier for programming because of the used languages (Visual Basic, Visual C#...), and the developing is much similar to developing on MS Windows platform. However, the phones that support it are not widely available in the local market, and are expensive.

Symbian OS smart phones, on the other hand, are widely available in the Middle East. There are two software platform based on Symbian OS; the S60 User Interface developed by Nokia for its smart phones, and the User Interface Quartz (UIQ) developed for Sony-Ericson, the later is based on touch screen while the S60 is based on navigation keys as an input device.

As the Nokia 6630 smart phone was chosen, the S60 UI platform will be used.

3.3.3.2 Programming Language

Two widely used programming languages for Symbian OS are considered.

The J2ME (Java 2 Micro Edition) is most appropriate for developing applications that do not need to interact with the low level functions of the phone (such as Bluetooth) and is easier in general. On the other hand, the Symbian C++ gives much more power to access the low level routines as it is the language used to write the Symbian OS.

3.4 Design Realization Approach

The project design will be implemented in hardware and software; the project will be installed in a real car and a smart phone, where evaluation and testing are to be applied.

3.5 Project Design Block Diagram

Following is the general block diagram of the system. A mobile phone uses Bluetooth to communicate with a control unit installed in the car; the control unit interacts with the car to perform the required operation.

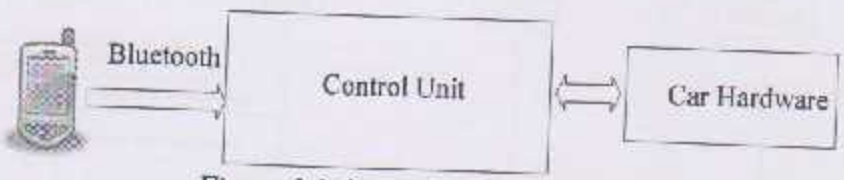

Figure 3.9 General Block Diagram

The block diagram of the control unit consists of three main parts; a Bluetooth module connected serially to a microcontroller, a microcontroller used to process received data, and a circuit connected to the microcontroller that will carry out the actions.

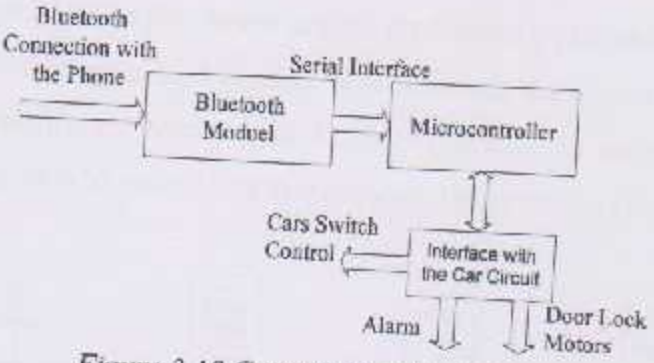

Figure 3.10 Control Unit Block Diagram

The control unit block diagram consists of three parts; a Bluetooth module for connecting with the phone, a microcontroller for processing messages from the phone and determine the action to be performed, and a circuit for interface the microcontroller with the car hardware.

3.6 System Modeling

3.6.1 Use Cases

Two use cases are related to the system:

- The car owner launches the remote control application on the mobile phone, provides the password for logging in, and chooses to lock/unlock the door, the phone sends the corresponding code to control unit that carry out the operation (Figure 3.12).

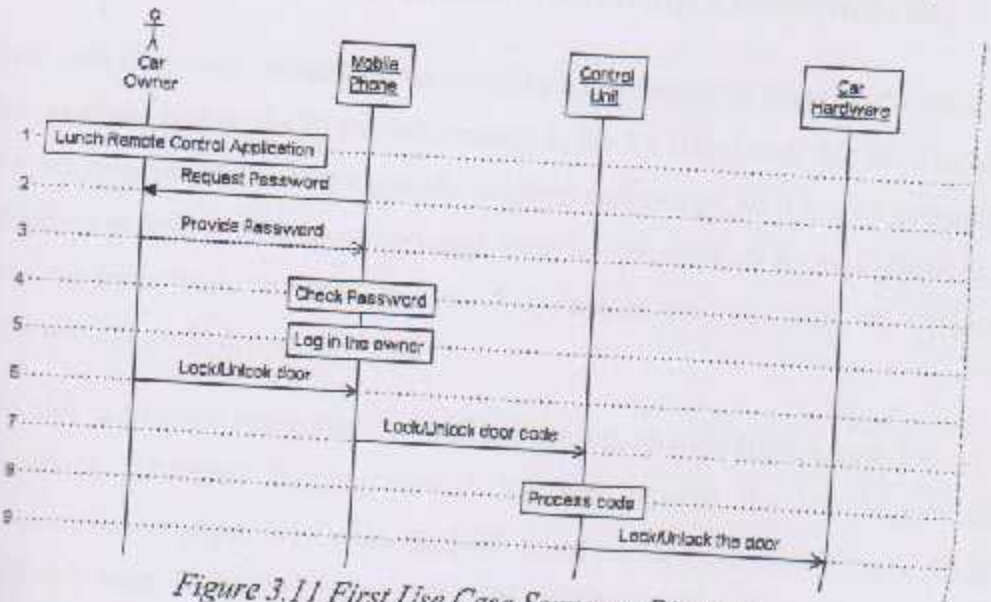

Figure 3.11 First Use Case Sequence Diagram

- The car owner launches the remote control application on the mobile phone, provides the password for logging in, and chooses to lock/unlock the switch, then provides the PIN (Personal Identification Number) for locking/unlocking the switch, the phone sends the corresponding code to control unit that carry out the operation (Figure 3.13).

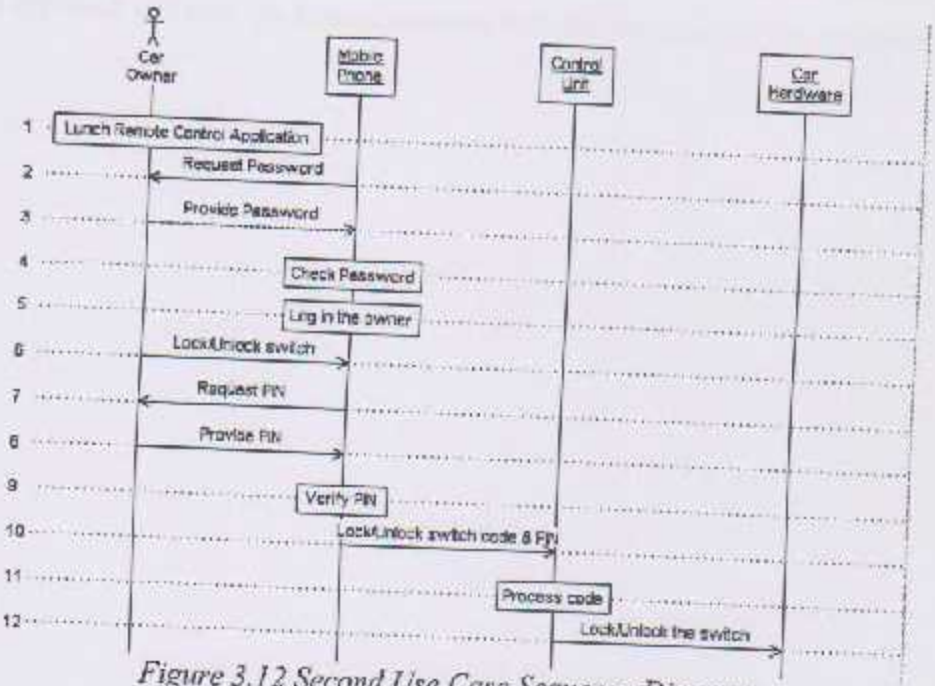

Figure 3.12 Second Use Case Sequence Diagram

3.7 Project Interaction with the Surrounding Environment

This project uses Bluetooth technology for wireless communication between the remote control and the car, as stated earlier; the Bluetooth operates in the 2.4 GHz band, this band is unlicensed; this means that it is not reserved for a specific wireless technology, so it's used by many devices such as microwave ovens, cordless phones, and some remote controls for toys. Devices working in this band are more likely to share the same frequency at the same time, so a technology is needed to detect and avoid this interference.

Fortunately, this is handled automatically by the Bluetooth module (inside both the car and the mobile phone). A Frequency Hopping Spread Spectrum is used, the two devices using the Bluetooth connection changes the frequency 1600 times per second; each time, a frequency is selected randomly form 79 frequencies in the 2.4 GHz band.

3.8 Summary

In this chapter we provided a detailed project objectives, discussed the design options of the system and its components and reasoned why we choose a specific option, the block diagram of the system was given, the data flow and use case models were shown, and finally described the realization approach and how the system interacts with the surrounding environment.

Chapter Four

Detailed Technical Project Design

4.1 Overview

4.2 Detailed Description of Project Phases

4.3 Subsystem Detailed Design

4.4 Overall System Design

4.5 User System Interface

4.6 Summary

4.1 Overview

In this chapter a detailed description of the main three phases (input, processing and output) is given for the project as a whole. Then a description of the subsystem detailed design and schematics. The complete system schematic is given next with a brief description of the system design. And lastly a view of the system user interface.

4.2 Detailed Description of Project Phases

Figure 4.1 shows the main phases of the system:

- The input to the system could be of two types: a sensor signal from the door (door is open or close) and switch (switch is on or off), and a command from the user transmitted wirelessly by the phone via Bluetooth. The sensor signals are transferred to the microcontroller via an optocoupler connected directly to the input port of the microcontroller, and the commands via Bluetooth module connected to the serial port.

- The processing is done at the microcontroller that takes actions with regard to the sensor signals and the user commands.

- Output of the system goes to the central locking motor in the main door, to the switch, and to the flashers. The output of the microcontroller is control signals that drive a number of relays which will carry out the desired operation.

- The power needed for the system is provided through the car 12V DC battery, most of the system components needs no more than 5V DC, so a voltage regulator is needed. The voltage regulator provides power to many components in the system. The Bluetooth module needs a special 3V regulation circuit.

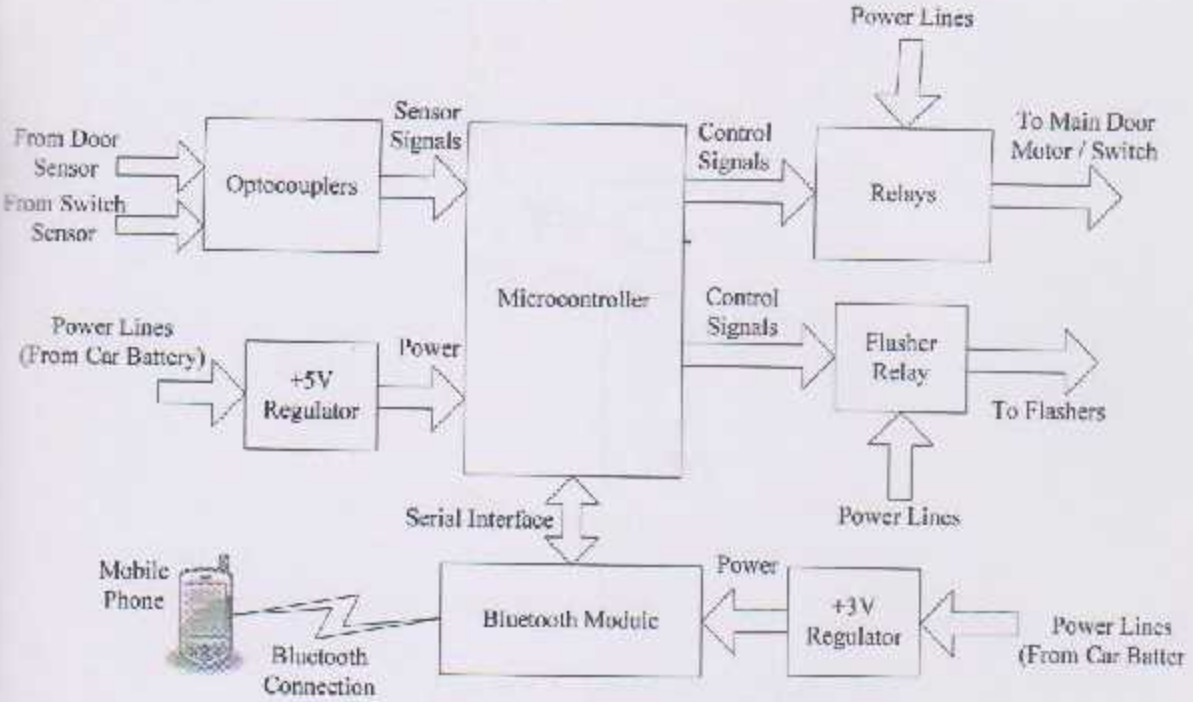

Figure 4.1 Detailed Block Diagram

4.3 Subsystem Detailed Design

4.3.1 Optocoupler

Because the signal coming from the door/switch sensor is 12V DC, and the microcontroller cannot hold this voltage, an optocoupler (aka phototransistor) is needed to transfer the signal to the microcontroller but keeping it isolated from the sensors.

The circuit for the switch sensor is the same as the figure below.

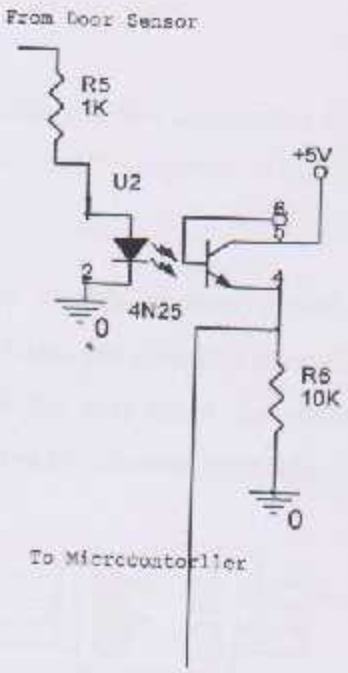

Figure 4.2 Optocoupler Circuit

4.3.2 Voltage Regulator

As the voltage source for the system is 12V DC, and many of the electronic components in the system needs only 5V, a 5V voltage regulator is required to supply the needed power.

The Bluetooth module needs a 3V power source; an adjustable voltage regulator is used to achieve this.

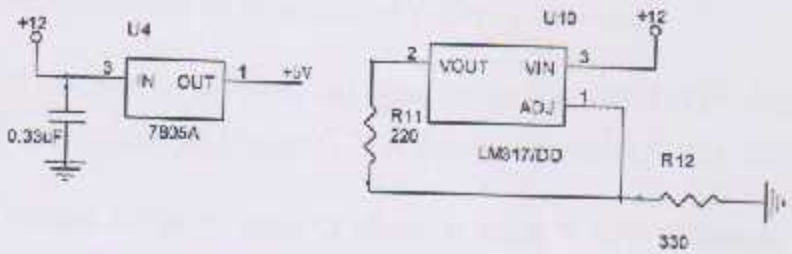

Figure 4.3 Voltage Regulator (5V left – 3V right)

4.3.3 Bluetooth Module

Bluetooth module uses the Bluetooth wireless connection to communicate with the phone, and the Universal Asynchronous Receiver/Transmitter (UART) to communicate with the host microcontroller.

The RXD and TXD pins of the module connect directly to TXD and RXD pins of the microcontroller respectively. The Status pin goes low when the Bluetooth connection exists with the phone, a LED is used to tell the user about the status of the connection. A 3 voltage regulation circuit is required to power the Bluetooth module.

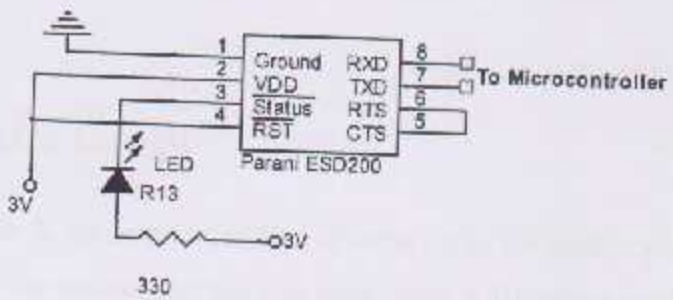

Figure 4.4 Bluetooth Module

4.3.4 Relays

The central locking motor, the flashers, and the switch need a +12V power source, as the microcontroller output supplies no more than +5V, a relay is needed.

A relay is an electrical switch that opens and closes under the control of another electrical circuit. The output of the microcontroller is used to drive the control circuit, which will drive the relay.

For the central locking motor to open or close, it needs a 12V difference across its sides, reversing the polarity reverses the state (from close to open or vice versa), to achieve this, two relays are used, the microcontroller closes one and opens the other to operate the motor, or closes them both to preserve the current state of the door (remain opened or closed).

The 74LS07 Buffer/Driver is used to drive the relay by providing the necessary current.

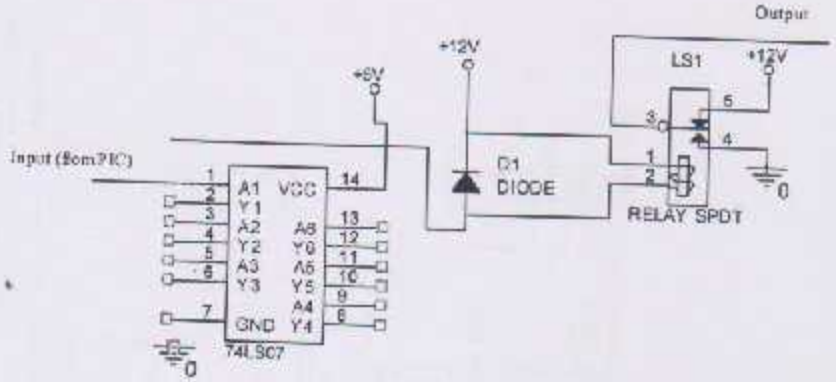

Figure 4.5 Relay Circuit

4.4 Overall System Design

As described in chapter 3, the system consists of three parts; the mobile phone, the control unit, and the car hardware. The control unit has two main parts; A Bluetooth module (Parani ESD200) to communicate with the phone, and a microcontroller (PIC18F4550) to process data and control the circuit.

The microcontroller interacts with the car hardware via relays which are driven by a hex driver (output), and with the signals coming from the car via optocouplers (input).

The Bluetooth module is interfaced with the microcontroller via the USART, with the following configuration: 9600 baud rate, 1 stop bit, and no parity. It is configured to only receive connections from the used mobile phone and to be in hidden mode where no other Bluetooth device could discover it.

The complete schematic for the project is shown below in figure 4.6

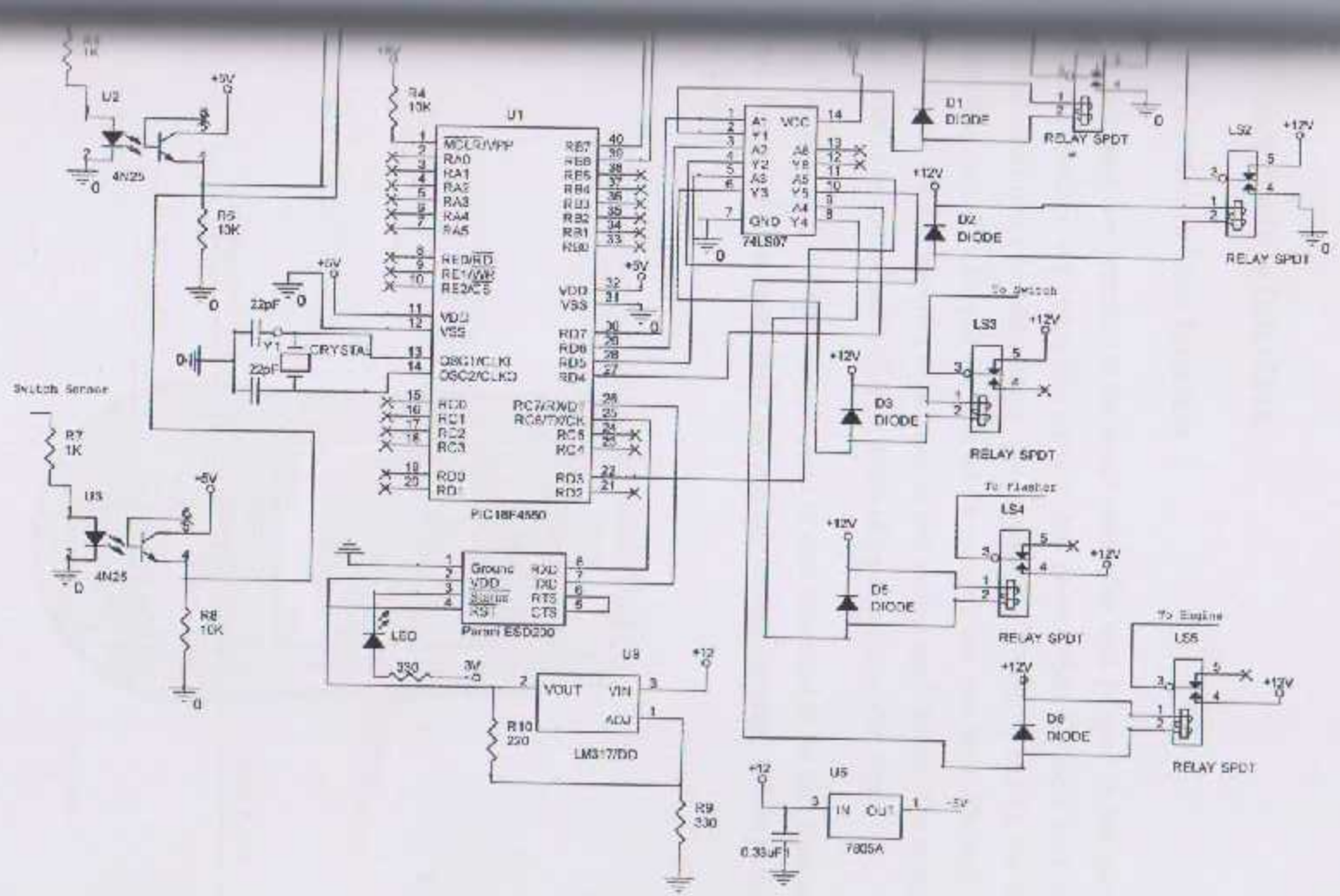

Figure 4.6 System Schematic

4.5 User System Interface

4.5.1 Hardware User Interface

The hardware user interface is the smart phone that will be used to navigate and control the software interface. All Nokia S60 series mobile phones share the same keys, these keys can be broken down into three subsections: the keypad (0 – 9 keys, *, and #), the soft keys, and the navigation keys. The keypad can be used for numeric and text input. The soft keys can be used for opening menus, selecting menu options and exiting applications. The navigation key allows the cursor to be moved and a click on it acts like a selection or OK press.

The user uses 1, 2, 3, 4 and 5 keypad keys to send command to the car; right soft key is used to send the application to background, left soft key for options menu, and the navigation to navigate the options.

Figure 4.7 Smart Phone Interface

4.5.2 Software User Interface

The GUI of the system which the user will interact with is kept simple so interaction becomes easier, as the user doesn't want to see a lot of dialogs and menus in order to perform a simple action as opening the door.

The GUI adapts the ordinary car remote control way of interacting, where the user has to press a key to perform an action. Keypad keys 1 – 5 perform actions, right soft key sends the application in background so the user can use other applications, and the left soft key opens the options menu where the user can exit the application.

When launching the application the user should provide a password in order to use the program.

Figure 4.8 Login Dialog

The following figure shows the main screen for the application, where the user needs to press a keypad key 1, 2, 3, 4 or 5 to perform an action.

Figure 4.9 Main Screen

The 'Unlock Switch' choice brings up a dialog for PIN input.

Figure 4.10 PIN Input Dialog

4.6 Summary

In this chapter the detailed description of the project phases was discussed, then the description of the characteristics and specifications of the subsystem design, next the overall system design schematics was given, and lastly a look at the hardware and software user interface.

Chapter Five

Software System Design

5.1 Overview

5.2 Software Needed for the Project

5.3 Flowcharts

5.4 Used Packages

5.5 Summary

5.1 Introduction

This chapter gives a description of the design of the microcontroller program, the program which will receive and decode the message send from mobile phone, send command to the Bluetooth module, and also will initiates commands to the car's center locking system hardware.

The program will communicate with a Bluetooth module and with electronic circuits that interfaces with the cars hardware.

Another important part of the software is the mobile phone software; that will be used to communicate with the system installed in the car.

5.2 Software Needed for the Project

Microcontroller Software Requirements

The program for microcontroller used in the system (PIC18F4550) will be written in the C programming language using a compiler developed for MS Windows XP operating system.
Writing the compiled program into the microcontroller EPROM is done using the ProPic 2 programmer through WinPIC800 programming software.

Mobile Phone Software Requirements

The mobile phone is programmed in Symbian C++ programming language, the compiler and all the needed tools are developed for MS Windows XP operating system. Installing the program on the phone is done using Nokia PC Suit form Nokia through a USB cable.

5.3 Flowcharts

Below are the flow chart for the control unit code and for the mobile phone remote control program and their explanation.

5.3.1 Microcontroller Program

Figure 5.1 shows how the microcontroller program works; when starts, it initializes the system (defines input/output ports, configures interrupt…). The microcontroller then waits for any data to process, when data is available the microcontroller checks wither its source is Bluetooth module (a command from the mobile phone) or from the car (a signal from switch, door…) and perform the needed action. When the operation completes, the microcontroller waits for data again.

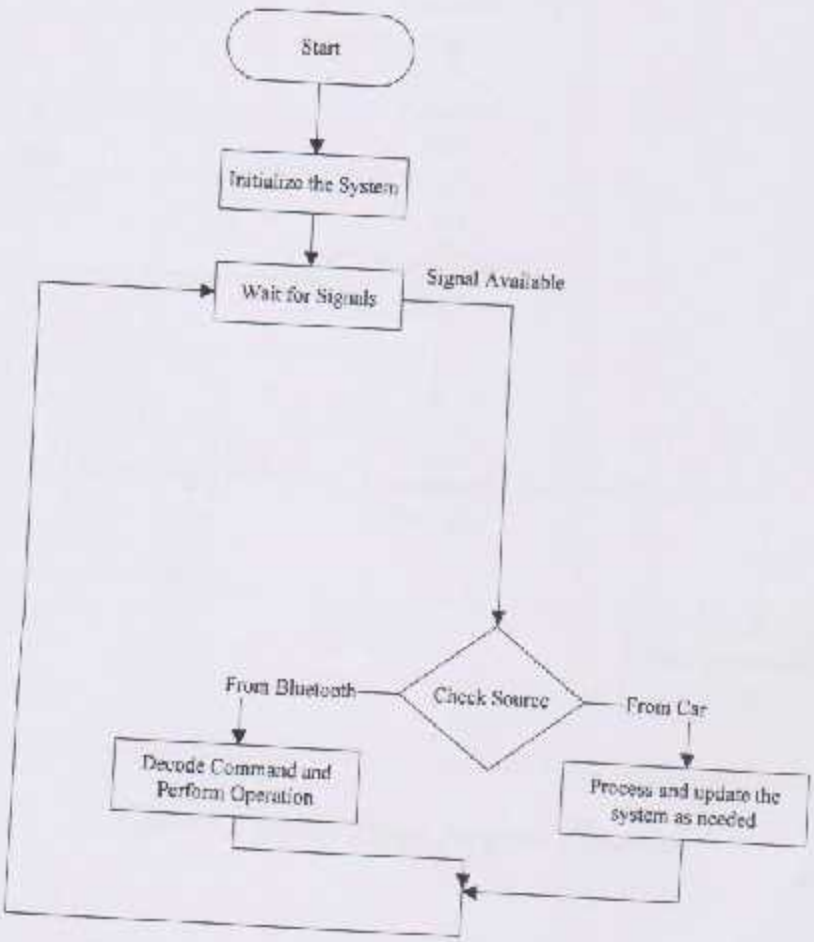

Figure 5.1 Microcontroller Program Flowchart

5.3.2 Smart Phone Program

In the mobile phone, for the program to work it requests a password when the system starts, if the password is not correct the program exits. Otherwise, the program displays a set of commands for the user to choose, when the user chooses a command, the program tries to connect and sends the command, if failed to connect it notifies the user and waits for input again.

The mobile phone program flowchart is shown in figure 5.2.

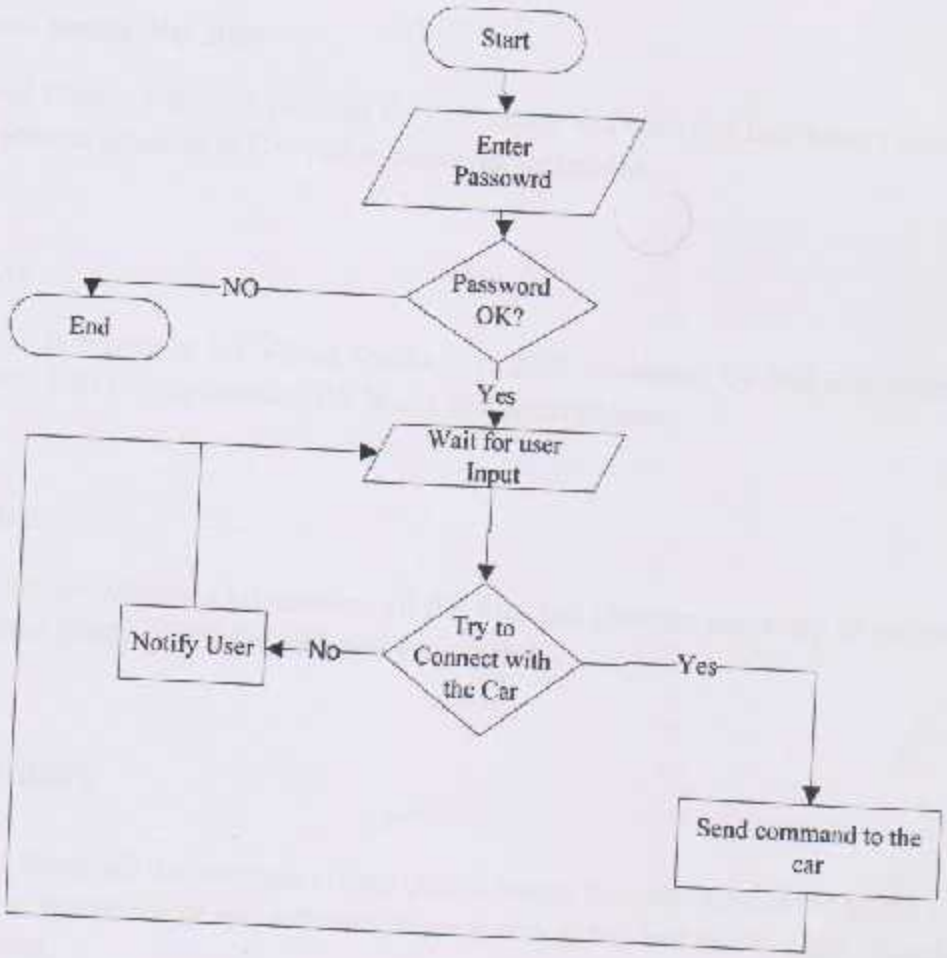

Figure 5.2 Mobile Phone Program Flowchart

5.4 Packages Used

A number of packages where used to develop the system:

MPLAB IDE and MPLAB C18

The MPLAB IDE is an integrated development environment from Microchip used to program the PIC microcontroller; it contains a text editor, compiler and a simulator. In order to program the PIC18F4550 in C, another package, the MPLB C18, is needed which contains the necessary C function for programming the PIC18F series.

MS Visual Studio .Net 2005

The Visual Studio .Net 2005 contains the C++ editor and compiler but doesn't support Symbian C++, in order to program in C++, other packages are needed.

Carbide.vc

Cardbdie.vc is a plug-in for Visual Studio .Net 2005 developed by Nokia in order to use the visual studio IDE in programming the Nokia S60 smart phones.

S60 2nd SDK

This software development kit contains all the files and libraries necessary to program the S60 series of smart phones, included with many examples.

5.5 Summary

This chapter discussed the software system design issues; the software needed by the project was described, the flowcharts of any software program was given, and finally a list of packages used in development.

Chapter Six

System Implementation and Testing

6.1 Overview

Testing is one of the most important phases when making any project, and is done throughout the project development to ensure the system will achieve its requirements and specifications. This chapter consists of two parts; in the first part describe how the system is implemented, the second part talks about testing the system and its components.

6.2 Actual Project Implementation

The project design has been implemented in hardware and software; in the hardware part, a central locking circuit that uses Bluetooth for communication was built with electronic components, in the software part, a program that make use of smart phone Bluetooth capabilities to send commands to the car is developed with the a programming language.

The hardware of the project (the central locking system) was built using PIC18F4550 microcontroller, a Bluetooth module, relays and other basic electronic components. These components were first put on a breadboard for testing, then on a printed circuit board (PCB). As a remote control unit, the Nokia 6630 smart phone was used.

Component	Used Number
PIC18F4550	1
Parani ESD200 Bluetooth	1
12V Relay	5
Optocoupler	2
Voltage Regulator	2
Hex Driver	1
Resistors, Capacitors, and LEDs	-
Nokia 6630 Smart Phone	1

Table 6.1 Components Used in the System

The software consists of two programs, a C program for the PIC18F4550 to control the central locking circuit and a Symbian C++ program for the smart phone to send commands via Bluetooth.

3 System Testing

3.1 Components Testing

o test circuits were built to test the two main components of the system; the microcontroller nd the Bluetooth module. Other basic parts (Relays, Hex drivers...) were not tested alone; nstead they were tested with the subsystem testing.

12V output DC adapter was used to provide power to system, a digital multi-meter to test oltages and currents, and a number of LEDs to check output stats.

3.1.1 PIC18F4550 Microcontroller

he Microcontroller plays an important role in the system, as it will carry out the received ommands and control the central locking circuit. In testing the used model (PIC18F4550), it as first programmed with the C language and used in a simple input output circuit where a DIP witch will turn on or off two LEDs. The testing circuit is shown in the figure below.

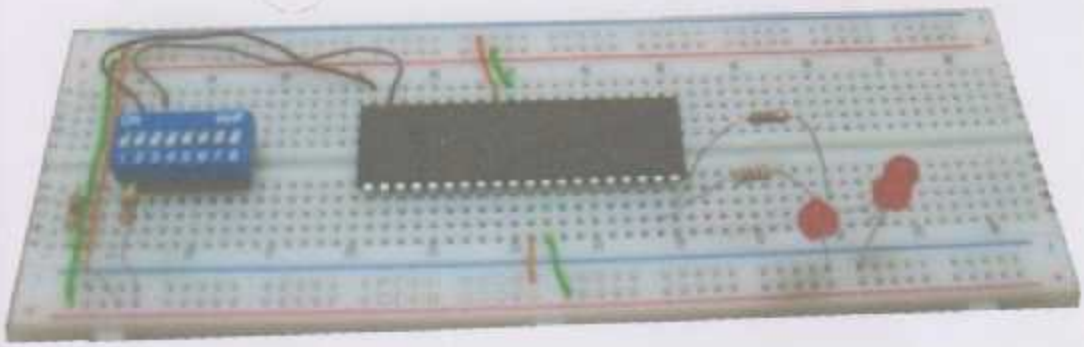

Figure 6.1 Microcontroller Testing Circuit

3.1.2 Parani ESD200 Bluetooth Module

he Bluetooth module is responsible for Bluetooth communication with the smart phone to eceive commands. It was tested by connecting it to the PC serial port via a MAX232 IC, then

g the hyper terminal program to send commands to it. A Bluetooth USB dongle connected to PC was used to connect to the Bluetooth module to test wireless connection.

3.2 Subsystem Testing

e system can be divided into two subsystems; the central locking subsystem and the Bluetooth system.

3.2.1 Central Locking Subsystem Testing

e central locking subsystem consists of the microcontroller, optocouplers, and relays. To mulate the input of Bluetooth a DIP switch is used. The switch is used to open/close the doors. is circuit was installed in the car were testing was done.

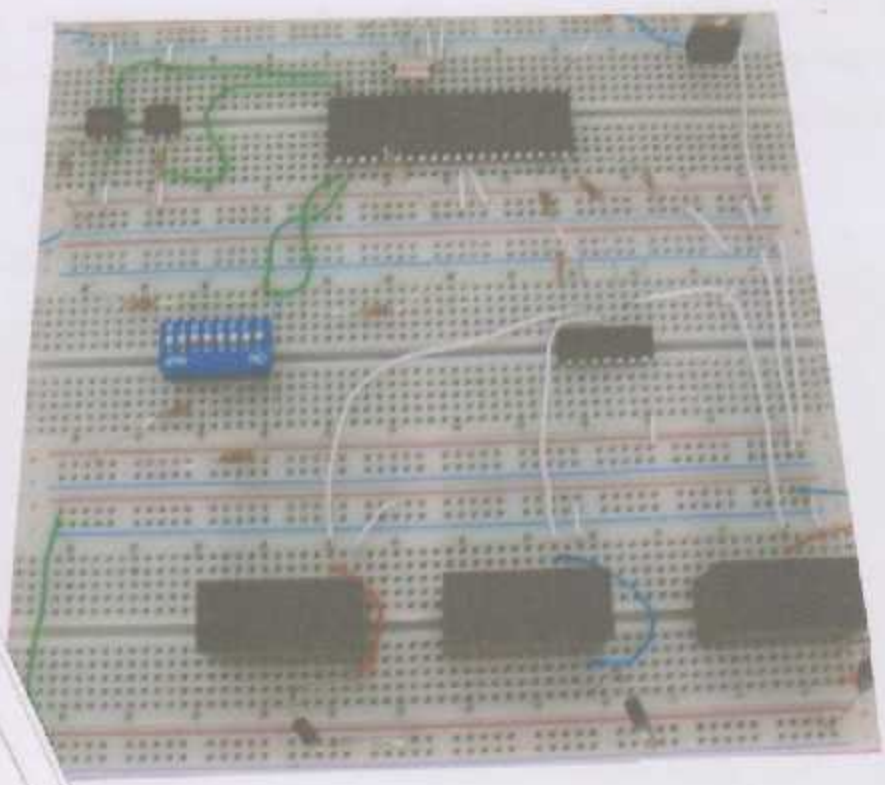

Figure 6.2 Central Locking Subsystem Testing Circuit

3.2.2 Bluetooth Subsystem Testing

The Bluetooth subsystem is the Bluetooth module connected to the microcontroller, for testing purposes, a MAX232 was needed to connect this subsystem with the PC for debugging. During this test, a problem was found when the Bluetooth sends a response message which is more than two bytes (an overflow occurs), making the microcontroller stuck when reading form the USART buffer, this was solved by clearing the overflow before reading.

6.3.3 Integrated Testing

The first integrated system test was done in the lab, the commands were sent using the Bluetooth dongle of the PC, and when the smart phone program was completed, it was used to send commands.

During this testing it was found that the microcontroller sometimes doesn't execute the received commands, although the Bluetooth module receives the command. After debugging and searching for solution the problem was because the Bluetooth module high output is 3V, and the microcontroller serial input reads this a low, the Bluetooth module was operating in its minimum voltage (3V), the voltage was raised to 3.3V which solved the problem.

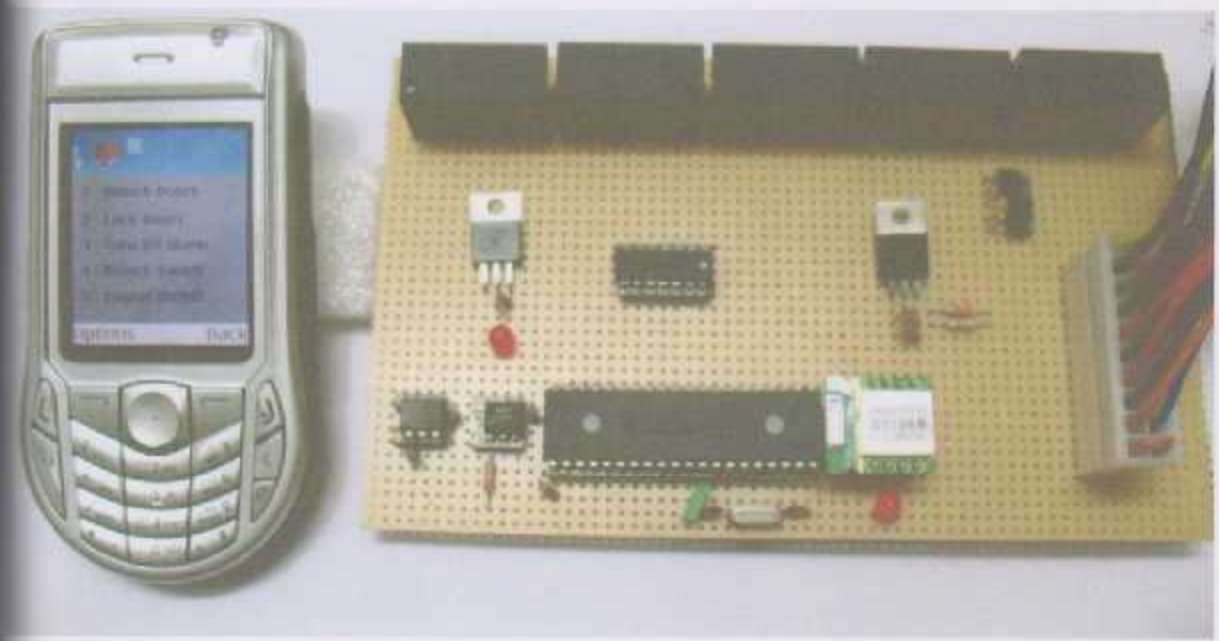

Figure 6.3 Complete System

The complete system was tested on two cars, Daewoo Matiz II and Hyundai Verna, and worked as expected. The range that the smart phone could send commands was about 20 meters without obstacles and about 10 meters with. All the operations of the system (lock/unlock door and switch, turn off the alarm, and turn engine on/off) worked as expected.

6.4 Summary

In this chapter the actual project implementation was discussed along with the various testing stages.

Chapter Seven

Conclusion and Future Work

7.1 Conclusion

7.2 Suggestions and Developments

7.1 Conclusion

While developing this project the team gains great experience in deferent areas in computer engineering: in the embedded systems development, in the electronics field, and in the smart phone development.

The final system achieved most of the design objectives except making the car send alarm messages to the smart phone, which couldn't be achieved because of the limitations in the Bluetooth module we had.

At the first times of the final testing the system failed to work in 40% of the attempts due to a problem when transferring the command to from the Bluetooth module to the microcontroller, after fixing that program, the system worked in more than 90% of the attempts, and was left working for four hours without failure.

7.1.1 Expected learning Outcomes

We were expecting to learn the following things before the beginning of implementing the project.

- Learn how to use and program the following :
    1. Program PIC microcontroller.
    2. Program S60 Nokia Smart Phone
    3. Use Bluetooth modules to send and receive data
- How to remotely control devices via smart phone.
- Learn how to control the car central locking and security system.
- Design a system which controls the door, switch, and window of the car.

7.1.2 Real Learning Outcomes

After the implementation of the project we have an expert in the following things:

- Learn how to use and program PIC18F4550 microcontroller.
- Learn how program Nokia S60 smart phone.
- How to remotely control devices via smart phone.
- Learn how to control the car central locking and security system.
- Controlling the central locking and security system of the car.
- Receiving data via Bluetooth from a smart phone, we were expecting to send alarm via Bluetooth, but we faced problems in making 2 way data communication between the Bluetooth module and the smart phone because the smart phone lacked full support for Bluetooth profile that was on the module.
- Designing a complete system that uses a smart phone to do the following with the car: lock/unlock doors, lock/unlock switch, and turn on/off the engine.

7.2 Suggestions and Developments

This project uses the smart phone as remote control for the car that does as the ordinary remote control plus the ability to lock/unlock the switch of the car with a personal identification number. Other developments for the project are:

- Sending feedback from the car to the phone (alarms for example).
- Allowing the smart phone to take full control of the car (driving the car).

Making a Bluetooth enabled car opens the way for many ideas in security and automation fields:

- A car security system that uses a Bluetooth module with external antenna (ranges about half kilometer) to transmit information (voice for example) from the car to a smart phone or a specially built receiver.
- A smart home where a number of Bluetooth modules installed in, when the car is in the range with the home Bluetooth, the garage door is opened, the lights around the home are put on…

References

[1] Mirtcho Spassov - Final Year Project - Yale University - spring 2005

[2] Richard Hoptroff - Elektor Electronics - summer 2004

[3] National Research Development Corporation - http://www.nrdcindia.com

[4]Thomas Fuhrmann, Markus Klein, and Manuel Odendahl, Karlsruhe University, Germany

[5] What is Smart Phone, Jo Best - <http://www.silicon.com>

[6] SymbianOS specifications - <http://www.symbian.com>

[7] Steve Babin - Developing Software for Symbian OS – Wiley - 2006

[8] Symbian OS - Wikipedia - <http://en.wikipedia.org/wiki/Symbian_os>

[9] Nokia S60 - Wikipedia - <http://en.wikipedia.org/wiki/Nokia_S60>

[10] Wireless Networking - Wikipedia

<http://en.wikipedia.org/wiki/Wireless_networking>

[11] Bluetooth SIG (2007) - Bluetooth Specification Version 2.1 –

<http://www.bluetooth.com>

[12] Bluetooth Whitepaper 1.1 - AU System - 2000

[13] Tom Denton - Automobile Electrical and Electronics - 2nd edition - Butterworth-Heinemann - 2000

[14] KC-21 Datasheet

[15] Microchip - <http://www.microchip.com>

Appendix A

Microcontroller Program

```
/* main.h */
 #include <p18f4550.h>
#include <usart.h>
#include <delays.h>
#include <string.h>

#pragma config FOSC = XT_XT
#pragma config FCMEN = OFF
#pragma config ICPRT = OFF
#pragma config PWRT = OFF
#pragma config BOR = OFF
#pragma config WDT = OFF
#pragma config CPUDIV = OSC1_PLL2
#pragma config PLLDIV = 1
#pragma config VREGEN = OFF
#pragma config XINST = OFF
#pragma config PBADEN = OFF
#pragma config DEBUG = OFF

#define SWITCH PORTBbits.RB7
#define DOOR   PORTBbits.RB6

void InterruptHandlerHigh (void);
void giveSound(void);


/* Main.c
B7+B6 as sensor of switch and door
D6+D7 for door lock control
D5 for switch controler
D4 Alarm And Flasher controler
D3 Engine off/on
*/

#include "main.h"

unsigned char  sFlag;  //switch flag
unsigned char  sFlag2; //switch flag
unsigned char  cFlag;  //last state of the door flag
unsigned char  eFlag;  //Flag to check engine state

void giveSound(void)
{
        LATDbits.LATD4=0;
        Delay10KTCYx(50);
        LATDbits.LATD4=1;
        Delay10KTCYx(80);
        LATDbits.LATD4=0;
        Delay10KTCYx(50);
        LATDbits.LATD4=1;
}

void main (void)
{
char temp1[8];
char temp2[26];
char temp3[18];
```

```
unsigned char c;

TRISB = 0xC0;                   //RB7 & RB6 Inputs
INTCON = 0x08;                 //Enable Intrrupt On Change
INTCON2 = 0x81;                // Intrrupt On Change high priority
RCONbits.IPEN = 1;             //enable priority levels
INTCONbits.GIEH = 1;           //enable interrupts

TRISD = 0x00;
TRISA = 0x00;
TRISC = 0b11000000;
RCSTAbits.SPEN = 1;

OpenUSART (USART_TX_INT_OFF &
           USART_RX_INT_OFF &
           USART_ASYNCH_MODE &
           USART_EIGHT_BIT &
           USART_CONT_RX &
           USART_BRGH_HIGH, 25);

sFlag = 1;
eFlag2 = 1;
cFlag = 1;
eFlag = 0;

PORTD = 0xF8;

while(1)
{
getsUSART(temp1,6);
LATDbits.LATD0 = 1;
A1:getsUSART(temp2,24); //wait for connected message
LATDbits.LATD1 = 1;

while(DataRdyUSART())
     c = RCREG;

RCSTAbits.CREN = 0;
RCSTAbits.CREN = 1;

while(1)
     {
     while(!DataRdyUSART());
          c = ReadUSART(); //read byte from the phone

       WriteUSART(c);
         if(c == 'O') //open doors
           {

                LATDbits.LATD6 = 1;
                LATDbits.LATD7 = 0;
                LATDbits.LATD4 = 1;//shtdown alarm if on
                giveSound();
                LATDbits.LATD7 = 1;
                LATDbits.LATD6 = 1;
                cFlag = 0;
                getsUSART(temp3,14);
```

```
        goto A1;
}
else if(c == 'C') //close doors
        {
         if(SWITCH == 1) //switch is on
              {
                   giveSound();
              }
        else if (DOOR == 0)//switch off but door open
                   {
                   giveSound();
                   }
        else
              {
                   LATDbits.LATD6 = 0;
                   LATDbits.LATD7 = 1;
                  giveSound();
                   LATDbits.LATD6 = 1;
                   LATDbits.LATD7 = 1;
                   cFlag = 1;
              }
        getsUSART(temp3,14);
        goto A1;
        }

else if(c == 'S') // open the switch
        {
              LATDbits.LATD5 = 0;
              giveSound();
              sFlag = 0;
              getsUSART(temp3,14);
              goto A1;
        }

else if(c == 'A') //shutdown the alarm
        {
              LATDbits.LATD4 = 1;
              giveSound();
              getsUSART(temp3,14);
              goto A1;
        }

else if(c == 'E') //Engine On/Off
        {
        if(SWITCH == 0) //switch is off
                   giveSound();
        else if(eFlag == 0) //switch on, but engine off
              {
              LATDbits.LATD3 = 0;
              Delay10KTCYx(110);
              LATDbits.LATD3 = 1;
              giveSound();
              eFlag = 1;
              }
        else //Engine is on
              {
              giveSound();
```

```
                        LATDbits.LATD5 = 1;
                        eFlag = 0;
                        }

                  getsUSART(temp3,14);
                  goto A1;
                  }
      }
}
}

#pragma code InterruptVectorHigh = 0x08
void
InterruptVectorHigh (void)
{
  _asm
    goto InterruptHandlerHigh //jump to interrupt routine
  _endasm
}

// High priority interrupt routine
#pragma code
#pragma interrupt InterruptHandlerHigh

void
InterruptHandlerHigh ()
{
if(INTCONbits.RBIF)
      {
      if(PORTBbits.RB6 == 0)
                  {
                        if(cFlag == 1)
                              {
                                    LATDbits.LATD4 = 0;
                              }
                  }
      }

      if(sFlag == 0)
                  {
                        if( PORTBbits.RB7 == 0 )  //swtich is on
                              {
                                    sFlag2 = 0;
                              }
                  }

      if(sFlag2 == 0)
                  {
                        if( PORTBbits.RB7 == 1 ) //switch off
                              {
                                    LATDbits.LATD5 = 1;
                                    sFlag = 1;
                                    sFlag2 = 1;
                              }
                  }
INTCONbits.RBIF = 0;
}
```

Appendix B

Smart Phone Program

```
// RCAppUi.h
#ifndef RCAPPUI_H
#define RCAPPUI_H
#ifdef __WINS__
#pragma warning (disable : 4100)
#endif

// INCLUDES
#include <eikapp.h>
#include <eikdoc.h>
#include <e32std.h>
#include <coecntx.h>
#include <aknappui.h>


// FORWARD DECLARATIONS
class CRCContainer;


// CLASS DECLARATION

/**
* Application UI class.
* Provides support for the following features:
* - EIKON control architecture
*
*/
class CRCAppUi : public CAknAppUi
      {
      public: // // Constructors and destructor

            /**
            * EPOC default constructor.
            */
            void ConstructL();

            /**
            * Destructor.
            */
            ~CRCAppUi();

      public: // New functions

      public: // Functions from base classes

      private:
            // From MEikMenuObserver
            void DynInitMenuPaneL(TInt aResourceId,CEikMenuPane* aMenuPane);


      private:
            /**
            * From CEikAppUi, takes care of command handling.
            * @param aCommand command to be handled
            */
            void HandleCommandL(TInt aCommand);
            void SendToBackground();
```

```
        virtual TKeyResponse HandleKeyEventL(
              const TKeyEvent& aKeyEvent,TEventCode aType);

    private: //Data
        // mutable
         CRCContainer* iAppContainer;
         TUint iLastPort;

    };

#endif

// ***
// RCContainer.h
#ifndef RCCONTAINER_H
#define RCCONTAINER_H

// INCLUDES
#include <coecntrl.h>
#include "RC_caption.rsg"
#include "RC.rsg"
#include <AknQueryDialog.h>

#ifdef __WINS__
#pragma warning (disable: 4100)
#endif

#include <aknview.h>
#include <aknlists.h>
#include <btdefcommport.h>
#include <btmanclient.h>
#include <c32comm.h>



// FORWARD DECLARATIONS
class CEikLabel;        // for example labels

// CLASS DECLARATION

class CRCContainer : public CCoeControl, MCoeControlObserver
    {
    public: // Constructors and destructor

        /**
        * EPOC default constructor.
        * @param aRect Frame rectangle for container.
        */
        void ConstructL(const TRect& aRect);

        /**
        * Destructor.
        */
        ~CRCContainer();
```

```
    public:
    void SetNewDevice(const TBTDevAddr& aAddress, const TDesC& aName);
    void ToggleSecurity();
    void ComPort(TDes& aString) const;
    void SetComPort(TInt aPort);
    void SetProfile(TUUID aProfile);
    void ResetPortSettings(TUint aPort);
      void SendToPhone(TInt aCh);
      void InitConnection();

    public: // Functions from base classes

    private: // Functions from base classes

        /**
           * From CoeControl,SizeChanged.
           */
           void SizeChanged();

        /**
           * From CoeControl,CountComponentControls.
           */
           TInt CountComponentControls() const;

        /**
           * From CCoeControl,ComponentControl.
           */
           CCoeControl* ComponentControl(TInt aIndex) const;

        /**
           * From CCoeControl,Draw.
           */
           void Draw(const TRect& aRect) const;


           void HandleControlEventL(CCoeControl* aControl,TCoeEvent
aEventType);
           TKeyResponse  CRCContainer::OfferKeyEventL (const TKeyEvent&
aKeyEvent,TEventCode aType);

        void GetSettingsFromRegistry();
        void ReadSettingsFromRegistry(TBTCommPortSettings& aSettings, TUint
aPort)const;
           void UpdateSettingsToRegistry();



    private: //data

        TBTCommPortSettings iSettings;
        RBTRegServ iRegistryServer;
        mutable RBTCommPortSettings iGlobalSettings;


        TUint iComPort;
```

```
        CEikLabel* iLabel1;
        CEikLabel* iLabel2;
        CEikLabel* iLabel3;
        CEikLabel* iLabel4;
        CEikLabel* iLabel5;

    };

#endif


// RCAppUI.cpp
// INCLUDE FILES
#include "RCAppui.h"
#include "RCContainer.h"
#include <RC.rsg>
#include "RC.hrh"

#include <avkon.hrh>
#include <aknnotewrappers.h>

#include <btcomcontrol.rsg>
#include "RC.hrh"
#include <bt_sock.h>
#include <btsdp.h>
#include <btextnotifiers.h>
#include <c32comm.h>

#include <apgtask.h>
#include <W32STD.H>
// ================= MEMBER FUNCTIONS =======================
//
// ----------------------------------------------------------
// CRCAppUi::ConstructL()
//
// ----------------------------------------------------------
//



void CRCAppUi::ConstructL()
{
    BaseConstructL();

    iAppContainer = new (ELeave) CRCContainer;
    iAppContainer->SetMopParent( this );
    iAppContainer->ConstructL( ClientRect() );
    AddToStackL( iAppContainer );
    iAppContainer->SetComPort(1);
    TInt64 addr64 (0x00000001 , 0x9507106B);
    TBTDevAddr devAddr1(addr64);



    //------
```

```
// Show dialog 'Enter password:'
      _LIT(pass,"adg");//adgjmptw
      TBuf<8> input;    // Max. length: 8
      CAknTextQueryDialog* dialog = CAknTextQueryDialog::NewL(input);
      if (dialog->ExecuteLD(PASS_Q ))
      {
            if(input.Compare(pass) !=0)
            {
          CAknInformationNote* note = new (ELeave) CAknInformationNote;
            note->ExecuteLD(_L("Wrong"));
            Exit();
            }
      }

      else
            Exit();

      iAppContainer->SetNewDevice(devAddr1,_L("MyCar-C7106B"));

      }

CRCAppUi::~CRCAppUi()
      {
      if (iAppContainer)
            {
            RemoveFromStack( iAppContainer );
            delete iAppContainer;
            }
   }
void CRCAppUi::DynInitMenuPaneL(
      TInt /*aResourceId*/,CEikMenuPane* /*aMenuPane*/)
      {
      }

TKeyResponse CRCAppUi::HandleKeyEventL(
      const TKeyEvent& /*aKeyEvent*/,TEventCode /*aType*/)
      {
      return EKeyWasNotConsumed;
      }

void CRCAppUi::HandleCommandL(TInt aCommand)
      {
      switch ( aCommand )
            {
            case EAknSoftkeyBack:
                  {
                  SendToBackground() ;
                  break;
                  }
            case EEikCmdExit:
                  {
                  Exit();
                  break;
                  }
            case   ERCCmdAppSelectDevice:
                  {
```

```
                    RNotifier not1;
                User::LeaveIfError(not1.Connect());

            TBTDeviceSelectionParamsPckg selectionFilter;
            TBTDeviceResponseParamsPckg response;
            TRequestStatus status;

            //Find only devices that support serial port profile
            selectionFilter().SetUUID(KSerialPortUUID);


not1.StartNotifierAndGetResponse(status,KDeviceSelectionNotifierUid,
                selectionFilter,response);
            User::WaitForRequest(status);

            if(KErrNone==status.Int())
                {
                if(response().IsValidBDAddr())
                    {
                    RDebug::Print(_L("[%02x%02x%02x%02x%02x%02x]"), \
                        response().BDAddr()[0], response().BDAddr()[1], \
                        response().BDAddr()[2], response().BDAddr()[3], \
                        response().BDAddr()[4], response().BDAddr()[5]);
                            if(response().IsValidDeviceName())
                                {
                                RDebug::Print(response().DeviceName());
                                iAppContainer-
>SetNewDevice(response().BDAddr(), \
                            response().DeviceName());
                                }
                            else
                                {
                        TBuf<16> name;
                        name.Format(_L("[%02x%02x%02x%02x%02x%02x]"),
                            response().BDAddr()[0], response().BDAddr()[1], \
                            response().BDAddr()[2], response().BDAddr()[3], \
                            response().BDAddr()[4], response().BDAddr()[5]);
                        iAppContainer->SetNewDevice(response().BDAddr(),
name);
                                }
                    }
                }
            not1.Close();

                break;
                }

                // TODO: Add Your command handling code here

            default:
                break;
            }
        }

void CRCAppUi::SendToBackground()
{
```

```
    // Construct an empty TApaTask object
    // giving it a reference to the Window Server session
    TApaTask task(iEikonEnv->WsSession( ));

    // Initialise the object with the window group id of
    // our application (so that it represent our app)
    task.SetWgId(CEikonEnv::Static()->RootWin().Identifier());

    // Request window server to bring our application
    // to background
    task.SendToBackground();}
// RCContainer.cpp
// INCLUDE FILES
#include "RCContainer.h"
#include <eiklabel.h>  // for example label control
#include <aknnotewrappers.h>
#include <btcommcontrol.rsg>
#include <btsdp.h>

_LIT(KPortFormatString, "BTCOMM::%d");

_LIT8(KOpen,  "O");
_LIT8(KClose, "C");
_LIT8(KAlarm, "A");
_LIT8(KSwitch,"S");
_LIT8(KEng,"E");

// ============================ MEMBER FUNCTIONS ===============================

// -----------------------------------------------------------------------------
// CRCContainer::ConstructL(const TRect& aRect)
// EPOC two phased constructor
// -----------------------------------------------------------------------------
//
void CRCContainer::ConstructL(const TRect& aRect)
    {
    CreateWindowL();

    iLabel1 = new (ELeave) CEikLabel;
    iLabel1->SetContainerWindowL( *this );
    iLabel1->SetTextL( _L("1 : Unlock Doors") );

    iLabel2 = new (ELeave) CEikLabel;
    iLabel2->SetContainerWindowL( *this );
    iLabel2->SetTextL( _L("2 : Lock Doors") );


    iLabel3 = new (ELeave) CEikLabel;
    iLabel3->SetContainerWindowL( *this );
    iLabel3->SetTextL( _L("3 : Turn Off Alarm") );


    iLabel4 = new (ELeave) CEikLabel;
    iLabel4->SetContainerWindowL( *this );
    iLabel4->SetTextL( _L("4 : Unlock Switch") );
```

```
	iLabel5 = new (ELeave) CEikLabel;
	iLabel5->SetContainerWindowL( *this );
	iLabel5->SetTextL( _L("5 : Engine On/Off") );

	User::LeaveIfError(iRegistryServer.Connect());
	User::LeaveIfError(iGlobalSettings.Open(iRegistryServer));
	GetSettingsFromRegistry();



	SetRect(aRect);
	ActivateL();
	}

// Destructor
CRCContainer::~CRCContainer()
	{
	delete iLabel1;
	delete iLabel2;
	delete iLabel3;
	delete iLabel4;
	delete iLabel5;



	iGlobalSettings.Close();
	iRegistryServer.Close();

	}

void CRCContainer::SizeChanged()
	{
	// TODO: Add here control resize code etc.
	iLabel1->SetExtent( TPoint(10,10), iLabel1->MinimumSize() );
	iLabel2->SetExtent( TPoint(10,37), iLabel2->MinimumSize() );
	iLabel3->SetExtent( TPoint(10,62), iLabel3->MinimumSize() );
	iLabel4->SetExtent( TPoint(10,87), iLabel4->MinimumSize() );
	iLabel5->SetExtent( TPoint(10,112), iLabel4->MinimumSize() );
	}

TInt CRCContainer::CountComponentControls() const
	{
	return 5; // return nbr of controls inside this container
	}

//
CCoeControl* CRCContainer::ComponentControl(TInt aIndex) const
	{
	switch ( aIndex )
		{
	case 0:
		return iLabel1;
	case 1:
		return iLabel2;
	case 2:
		return iLabel3;
	case 3:
```

```
        return iLabel4;

    case 4:
        return iLabel5;

    default:
            return NULL;
        }
    }

// ----------------------------------------------------
// CRCContainer::Draw(const TRect& aRect) const
// ----------------------------------------------------
//
void CRCContainer::Draw(const TRect& aRect) const
    {
    CWindowGc& gc = SystemGc();
    // TODO: Add your drawing code here
    gc.SetPenStyle( CGraphicsContext::ENullPen );
    gc.SetBrushColor( KRgbGray );
    gc.SetBrushStyle( CGraphicsContext::ESolidBrush );
    gc.DrawRect( aRect );
    }

TKeyResponse  CRCContainer::OfferKeyEventL (const TKeyEvent&
aKeyEvent,TEventCode aType)
{
    if (aType==EEventKey)
    {
            switch (aKeyEvent.iCode)
        {
            case '1':
            {
            SendToPhone(1);
        break;

            return EKeyWasConsumed;
            }

            case '2':
            {
            SendToPhone(2);

        break;

            return EKeyWasConsumed;
            }

            case '3':
            {

                SendToPhone(3);
        break;

            return EKeyWasConsumed;
            }
```

```
                    case '4':
                    {
                        // Show dialog 'Enter PIN:'
                        _LIT(PIN1,"123");
                        TBuf<8> input;    // Max. length: 8
                        CAknTextQueryDialog* dialog =
CAknTextQueryDialog::NewL(input);
                            if (dialog->ExecuteLD(PIN_Q)) {
                                if(input.Compare(PIN1) != 0)
                                {
(ELeave) CAknInformationNote;
                                    CAknInformationNote* note = new

                                    note->ExecuteLD(_L("Wrong PIN!"));
                                }
                                else
                                {
                                    SendToPhone(4);
                                }
                            }
                            else {
                                // Cancel
                            }

                        return EKeyWasConsumed;
                    }

                    case '5':
                    {

                        SendToPhone(5);
            break;

                    return EKeyWasConsumed;
                    }

                }
        }

        return EKeyWasNotConsumed;
}


void CRCContainer::HandleControlEventL(
    CCoeControl* /*aControl*/,TCoeEvent /*aEventType*/)
    {
    // TODO: Add your control event handler code here
    }

void CRCContainer::SetComPort(TInt aPort)
    {
    iComPort = aPort;
    iSettings.SetPort(iComPort);
    GetSettingsFromRegistry();
    }

void CRCContainer::SetNewDevice(const TBTDevAddr& aAddress,
    const TDesC& aName)
```

```
    {
    iSettings.SetBTAddr(aAddress);
    iSettings.SetName(aName);
    UpdateSettingsToRegistry();
    }

void CRCContainer::ToggleSecurity()
    {
    if(iSettings.IsEncryptionSet())
        {
        iSettings.SetNoSecurity();
        }
    else if(iSettings.IsAuthenticationOnlySet())
        {
        iSettings.SetAuthenticationAndEncryption();

        }
      else
        {
        iSettings.SetAuthentication();
        }
    UpdateSettingsToRegistry();
    }

void CRCContainer::GetSettingsFromRegistry()
    {
    iSettings.SetPort(iComPort);
    iSettings.SetName(KNullDesC); // work around stale data
    iGlobalSettings.Get(iSettings);

    }
void CRCContainer::UpdateSettingsToRegistry()
    {

    TRequestStatus status;
    iGlobalSettings.Modify(iSettings, status);
    User::WaitForRequest(status);

    }

void CRCContainer::SetProfile(TUUID aProfile)
    {
    iSettings.SetUUID(aProfile);
    UpdateSettingsToRegistry();
    }

void CRCContainer::ComPort(TDes& aString) const
    {
    aString.Format(KPortFormatString, iComPort);
    }


void CRCContainer::SendToPhone(TInt aCh)
    {
            TBuf<16> port;
        TInt err;
        RCommServ server;
```

```
        RComm comm;


              CleanupClosePushL(server);
        User::LeaveIfError(server.Connect());
        err=server.LoadCommModule(_L("BTCOMM"));
        if((err!= KErrNone) && (err!= KErrAlreadyExists))
            {
            User::Panic(_L("Load"), err);
            }
        ComPort(port);
        User::LeaveIfError(comm.Open(server, port,
ECommShared,ECommRoleDTE));

              TRequestStatus status;

        switch(aCh)
        {
        case 1:

      comm.Write(status, KOpen);
      User::WaitForRequest(status);

              break;
        case 2:

        comm.Write(status, KClose);
        User::WaitForRequest(status);

        break;
        case 3:

        comm.Write(status, KAlarm);
        User::WaitForRequest(status);

        break;

        case 4:
            comm.Write(status, KSwitch);
        User::WaitForRequest(status);
              break;

        case 5:
              comm.Write(status, KEng);
        User::WaitForRequest(status);

        break;
        }
         comm.Close();
    CleanupStack::PopAndDestroy(1);
}
```