Palestine Polytechnic University

Collage of Engineering
Electrical Engineering Department

Automatic Parking System (APS)

Prepared by:

Musab Alnajjar , Suha Abusharkh , Mohammad Al-daraweesh

Supervisor:

Eng. Ayman Wazwaz

Submitted to the College of Engineering
in partial fulfillment of the requirements for the degree of
Bachelor degree in Communication and Electronics Engineering
Palestine Polytechnic University

June 2015

جامعة بوليتكنك فلسطين
Palestine Polytechnic University
(PPU)
The Library المكتبة
Acc. 28848 الرقم المتسلسل
Class. رقم التصنيف

Palestine Polytechnic University
Collage of Engineering
Electrical Engineering Department

Project Name

Automatic Parking System (APS)

Project Team

Musab Alnajjar, Suha Abusharkh, Mohammad Al-daraweesh

According to the project supervisor and according to the agreement of the testing committee members, this project is submitted to the Department of Electrical Engineering Department at college of engineering in partial fulfillment of requirement of Bachelor degree in Communication and Electronics Engineering.

Supervisor signature

Eng. Ayman Wazwaz

.....

Department Head signature

.....

الإهداء

إلى من أعطاني اسمه ودمه ،، إلى المعلم الأول ،، وسيد الرجال ،، إلى من أضاء لي دروب العلم ،، إلى من لا يغفو قلبه إلا بعد أن تغفو جميع القلوب ،، إلى من أنا رجلٌ لأني جزء منه ،، إلى الجبل الشامخ ،،
إليك يا أبي.

إلى المدرسة الأولى ،، وصانعة الرجال ،، إلى الحبّ الأبديّ ،، إلى من أهدتني صحتها وشبابها ،، إلى العين الساهرة ،، إلى من تراني طفلها وإن كنتُ كهلا ،، إلى القلب الدافئ ،،
إليك يا أمي.

إلى الجسد الواحد ،، إلى من أشاركهم دمي ،، إلى ملاذي عند الشدائد ،، إلى من لا تكلّ أيديهم من مدها لي ،،

إليكم أخواتي و أصدقائي.

إلى الدرع المتين ،، إليكم يا نبع العلوم ،، إلى كل من علمني ،، وأخذ بيدي ،، وأنار لي طريق العلم والمعرفة
إليكم أساتذتي الأفاضل.

إلى بيتنا الثاني ،، إلى الصرح العلمي الشامخ ،، إلى الكنز الثمين ،،
إليك جامعتي.

إلى الأرض الطاهرة ،، وأرض الأنبياء ،، إلى الأرض التي باركها الله ،، إلى الصمود والشموخ والكبرياء ،،
إليكِ فلسطين.

Acknowledgement

Our thanks go first of all to our project supervisor Eng. Ayman Wazwaz, his guidance and support made this work possible.

We wish to thank Dr. Ramzi Qawasma, Eng. Sa'di Y. Tamimi, we believe that this work would not be accomplished without their inspiration.

Finally, our thanks go to all lecturers & doctors, engineers, and laboratory supervisors in PPU. Their effort and their nice dealing with us improved our characters to become successful engineers in the future.

Abstract

The increasing number of cars every year causes many problems, which affects the life of people everywhere, and increasing the need of more parks, and more time is needed to find a place for a car to park.

Our project proposes a solution to decrease the effect of this problem in our society. The idea of our project is to design an Automatic Parking System (APS) for drivers to reserve the suitable park slot for their cars before they arrive, which saves time, money and effort.

Our system is based on sensors to detect the available slot, Arduino to manage park slots, gates, and a ZigBee network to connect with a server. The server is connected with a modem to communicate with mobile devices and billing service.

List of Contents:

Dedication II

Acknowledgement III

Abstract IV

List of Contents V

List of Tables VIII

List of Figures IX

Important abbreviations XI

Chapter One: Introduction **1**

1.1 Overview 2

1.2 Objectives and Requirements 3

1.3 Motivations 4

1.4 Approach 4

1.5 Related Works 4

1.6 Project Plan 6

Chapter Two: Background **9**

2.1 ZigBee and other competitors 9

2.2 Keypad 12

2.3 Arduino 12

2.3.1 Why Arduino? 13

2.3.2 Types of Arduino 13

2.4 Global System for Mobile 15

2.5 Sensors 16

2.5.1 Photoelectric sensor 16

2.5.2 Ultrasonic sensor 17

2.6 Software 18

2.6.1 C# Programming Language 18

2.6.2 Java Programming Language 18

2.6.3 Microsoft Office Access 19

2.6.4 SQLite Database 20

2.6.5 Arduino Software 20

Chapter Three: System Design **23**

3.1 General Block Diagram 23

3.2 Billing 24

3.3 Data Base..... 24

3.4 Main System Components 25

3.4.1 Sensors 25

3.4.2 Arduino Mega 27

3.4.3 Keypad 33

3.4.4 ZigBee 33

Chapter Four: Software Requirement Specification **38**

4.1 Requirement Description 38

4.1.1 Actors 38

4.1.2 Usage Scenario 38

4.2 Flow Chart 41

4.2.1 Booking Flow Chart 41

4.2.2 Entry Flow Chart 42

4.2.3 Exit Flow Chart 43

4.2.4 Billing Flow Chart 44

4.3 Step by Step Scenario 45

Chapter Five : System Implementation **48**

5.1 Hardware system implementation 48

5.2 Software system implementation 52

5.2.1 Main Interface 52

5.2.2 Slots Interface 54

5.2.3 Reservation Interface 56

5.2.4 Message Interface 57

5.2.5 Data Flow Interface 58

5.3 Conclusion 59
5.4 Recommendations..... 60
Appendix A ..... A
Appendix B ..... B
References

List Of Tables:

Table Number	Description	Page
Chapter Two		
Table 2.1	Comparison between technologies	11
Chapter Three		
Table 3.1	The data base designed for the system	25
Table 3.2	Characteristics about Arduino Mega	28
Table 3.3	Datasheet for ZigBee	33
Chapter Four		
Table 4.1	Logging on and access the system	38
Table 4.2	Maintain the system	39
Table 4.3	Monitor the system	39
Table 4.4	Request Reservation	40
Table 4.5	Enter or exit car in certain slot	40

List Of Figure

Figure1.1 System Block Diagram 8
Figure 2.1: Mesh Networks 10
Figure 2.2 : Data Format 11
Figure 2.3 : ZigBee Examples 12
Figure 2.4 : Keypad 12
Figure 2.5 : Arduino Uno 14
Figure 2.6 : Arduino GSM Shield 14
Figure 2.7 : Arduino MEGA ADK 15
Figure 2.8 : Photoelectric sensor 16
Figure 2.9 : Ultrasonic sensor 17
Figure 2.10: C# Programming Language Logo 19
Figure 2.11: Java Programming Language Logo 19
Figure 2.12: Arduino Software 21
Figure 3.1: Block Diagram for the system 23
Figure 3.2: The block diagram of the Arduino work in the system 27
Figure 3.3: Arduino Mega module 28
Figure 3.4: The data sheet for Arduino Mega 29
Figure 3.5: Interface between Arduino and Ultrasonic sensor 31
Figure 3.6: A prototype for the subsystem of Arduino 32
Figure 3.7: Interface between Arduino and ZigBee 32
Figure 3.8 : Internal component of keypad 33
Figure 3.9 : The ZigBee 34
Figure 3.10 : Data packet as transmitted through the ZigBee 34
Figure 5.1 : General Block Diagram for Sub-Systems.....48
Figure 5.2 : USB-GSM modem, PC together.....49
Figure 5.3 : XBee, PC and the dongle all together.....49
Figure 5.4 : Circuit diagram of the coordinator node.....49
Figure 5.5 : Type of Messages.....50
Figure 5.6 : Type of Phones.....50

Figure 5.7 : The Reference Node.....51

Figure 5.8 : The Main Interface before connect XBee PRO Module and USB-GSM Modem..52

Figure 5.9 : The Main Interface after connect XBee PRO Module and USB-GSM Modem.....53

Figure 5.10 : Slots Interface and status of slots are free.....54

Figure 5.11 : Slots Interface and status of slot reserved.....55

Figure 5.12 : Reservation Interface.....56

Figure 5.13 : Message Interface.....57

Figure 5.14 : Data Flow Interface.....58

Important abbreviations

GSM : Global System for Mobile.

RFID: Radio Frequency Identification.

PDA: Personal Digital Assistant.

M2M Network: Machine to Machine Network.

RF: Radio Frequency.

DSSS: Direct Sequence Spread Spectrum.

AES: Advanced Encryption Standard.

I/O: Input / Output.

ADC Channel: Analog-to-Digital Converter Channel.

DC: Direct Current.

AC: Alternating Current.

PWM: Pulse Width Modulation.

USB: Universal Serial Bus.

ICSP header: In Circuit Serial Programming.

FTDI-USB: Future Technology Devices International-Universal Serial Bus.

SIM: Subscriber Identity Module.

GPRS: General Packet Radio Service.

SMS: Short Message Service.

TDMA: Time Division Multiple Access.

HSCSD: High-Speed Circuit-Switched Data .

EDGE: Enhanced Data GSM Environment.

UMTS: Universal Mobile Telecommunications Service.

JVM: Java Virtual Machine.

WORA: Write Once Run Anywhere.

PC: Personal Computer.

SRAM: Static Random Access Memory.

EEPROM: Electrically Erasable Programmable Read Only Memory.

MISO: Multiple Input Single Output.

SPI: Serial Peripheral Interface.

Chapter One

Chapter 1
Introduction

1.1 Overview

In the past, when driver want to visit any city by his car , driver should park his car in a safe park , and to do that by a traditional way there are many challenges facing driver such as : consuming more time , spent more money before reach a park . and this is not only when driver reach the park and want to buy a parking ticket and driver don't have enough money in your pocket.

Now, driver will be more comfortable from where should park his car, and don't worry about it. Our project allow to driver know where the nearest empty park to reserve it from far distance, then go directly to the park. To pay reserving fees the system will discount the required amount from driver phone balance that's more suitable than traditional billing or using credit card.

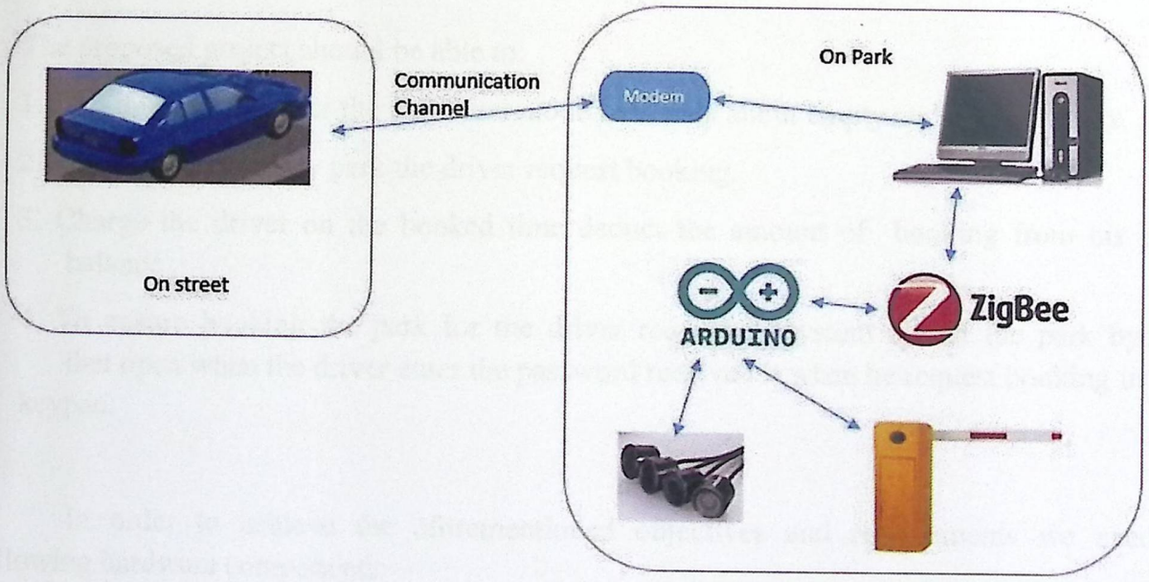


Figure1.1 System Block Diagram.

1.2 Objectives and Requirements

1.2.1 Design Objectives

The proposed project is assumed to achieve the following objectives:

• The system should be able to organize parks electronically to know the number of empty positions in every park.

•The system should enable the driver to know the nearest empty parking slot, to save time.

•The system should allow the driver to reserve slot for his car remotely and then go to the slot formerly reserved, so it saves the time required to search for slot and book it.

•The system should facilitate the driver to pay booking fees for parking, by rebating the required amount from the balance of Mobile driver.

•The system should enable the driver to follow the location of his car during the Camera Monitor on-site park and a confirmation of the safety car.

1.2.2 System Requirements

The proposed project should be able to:

1. Continuously monitor the parks and modify the data about empty and booked parks.
2. Reserving any empty park the driver request booking.
3. Charge the driver on the booked time deduct the amount of booking from his phone balance .
4. To ensure booking the park for the driver request it, system closed the park by gate, that open when the driver enter the password received it when he request booking using keypad.

In order to achieve the aforementioned objectives and requirements we need the following hardware components:

. ZigBee Transceivers : for wireless communication.

. Microcontrollers: to control devices

. Sensors: for monitor the state of the spaces at the parking and feedback the state.

. Server: to store data and run the software.

. Mobile : to do the reservation remotely.

.Gates : to reserving the park and to prevent anyone to fill the reserved park.

. Cameras : to monitor all parks. (Optional)

1.3 Motivations

- Reduction of car theft by monitoring park.
- Reduce traffic in the streets, by make operation of research easier to find a position to park car and faster.
- To maintain the beauty and cultural city by reducing the number of cars parked on both sides of the road.
- Reduce traffic violations resulting from the wrong stand in the streets.
- Save time and effort and money, by fixing the park for the driver, to head him directly without need to research for empty park.

1.4 Approach

1. Our system is proposed to solve spread parks on sides of street in Hebron city.

2. A collection of sensors should be distributed to find a suitable place for a car and reserve it when the driver request reserve.

3. Communication will be throughout mobiles (GSM). In this case the name of the park, the number of the empty park, and secret word to enable the driver to open the booked park.

4. This system should be attached with central information base in the municipality, to Supply it with all booking process. not only reserving process , but the road traffic and parks empty. All these aspects leads to help the planning process and directing transportation in future.

5. The billing process is as following: the driver should pay the booking fees for an hour, by rebate the required amount from the balance of Mobile driver

6. The driver has the right to be given a 20 minutes to get benefit from the booking " the first hour", after that period the booking will cancel, if the driver do not occupy park.

7. When the driver leaves the park he should pay the booking fees according to the number of booking hour.

8. When the driver arrive the park he hasn't the right to open the barrier unless he knows the right secret word, the gate open when the driver enter the right secret word using keypad, to allow him enter his car. He can't leave without paying the required amount .

1.5 Related Works

1.5.1 Smart Car Parking System

Smart Car Parking System, R.Zallom, J.Khamayseh, and M.Al-mohtaseb.

Graduation project 2007.

This project is involved in design and implementation of a smart parking system. The system will direct the driver to available parking spaces in each floor when entering the parking. Then, from the floor the driver will be derived to the parking available slots.

The project will be implemented with the 8051 microcontroller for all the necessary controlled hardware components are used such as sensors display and motor for the system.

This project give a guidance parking system.

Advantage:

- Very useful for controlling the parking hardware part.

Disadvantage:

- No payment system.

Different from our project in:

- No payment system.
- No identification.

1.5.2 Wireless Operated Parking System

Wireless Operated Parking System, Sh.Arafeh, Sh.Wazwaz, M.Al-juneidi.

Graduation project 2011.

This project was used to automate the parking process, by designing a system consists of sensors and counter that monitor the state of the spaces at the parking and feedback the administration center with data about the parking spaces, and this administration center communication with the driver and opens the gate if there is a space and if this driver is permitted to enter this parking. In other words the driver communicate with the administration center and ask it to enter the garage, the administration center check the identity of the driver and the park status, and then sends a signal to the gate to open.[1]

Advantage:

- Very useful for controlling the park .
- Payment system.

Disadvantage:

- When the Power failure all counters will reset and all data is volatile .

Different from our project in:

- No counters.
- Does not allow book from far distance.

1.5.3 Smart Parking System Using RFID Technology

Smart Parking System Using RFID Technology, A.Al-fouqha, M.Al-masri. Graduation project 2014.

The principal goal of the project was build a Radio Frequency Identification System (RFID) vehicle tracking device to provide a more user-friendly, affordable, and an efficient means to track vehicles accurately in a parking lot. Radio Frequency Identification (RFID) based parking entrance system becomes different and provides a more secure way to enter into the parking than traditional parking systems. As in traditional parking systems before entering into the parking the car is checked by the security guards, that whether a car belongs to that particular parking area or not, notes the vehicle registration number, and issued a slip or token due to which time is wasted and it still may not be the secure way.

In RFID based parking entrance system reader will detect the car, as the car will be 4-5m faraway from gate and guarantees whether the vehicles belongs to that parking area or not? This will ensure the entrance of only authorized vehicles, hence decreasing time consumed and ensuring high security. [2]

Advantage:

- Very useful for controlling the parking .
- Security system.

Disadvantage:

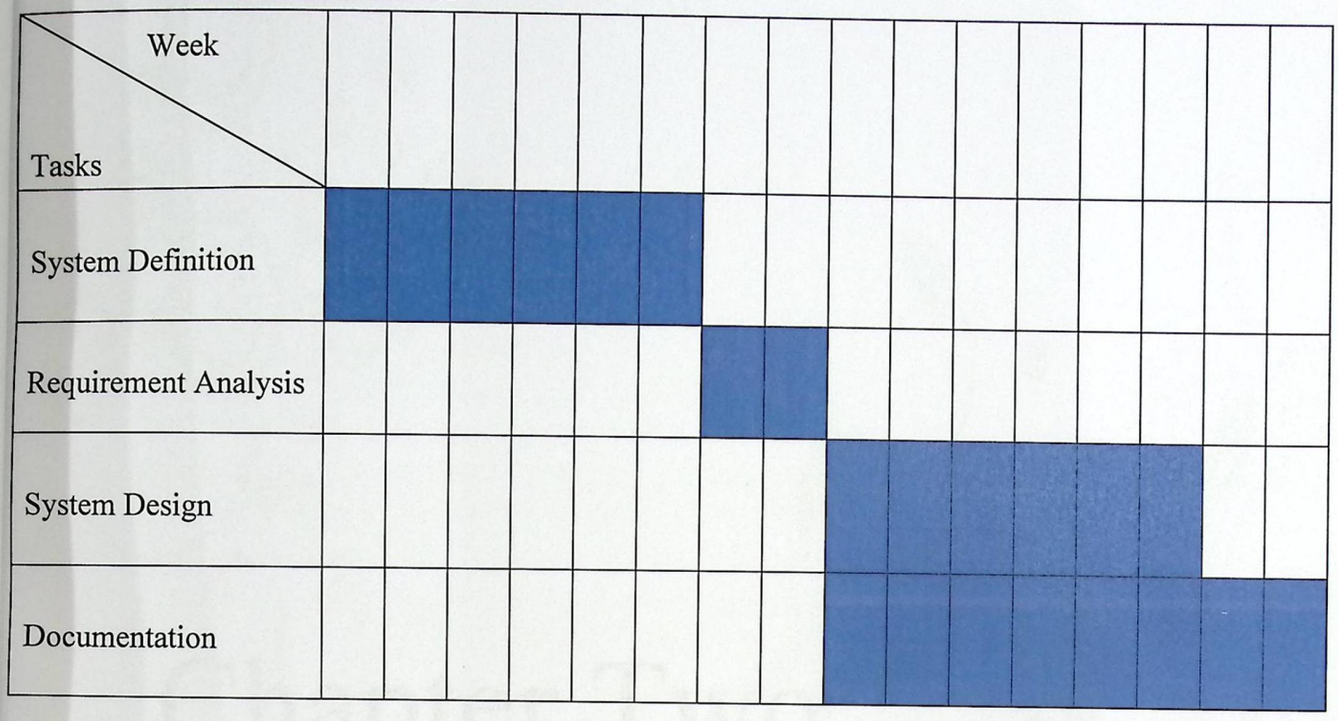
- No Payment system.

Different from our project in:

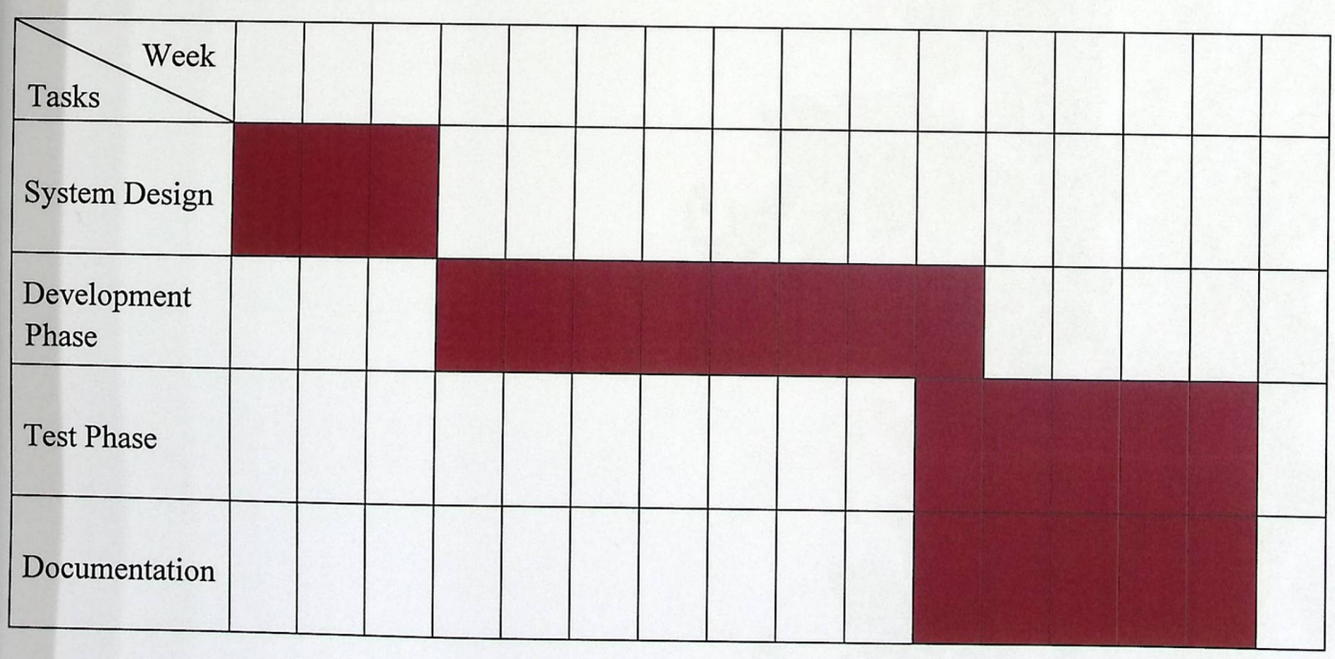
- No Payment system .
- Does not allow book from far distance.

1.6 Project Plan

1.6.1 First Semester



1.6.2 Second Semester



Chapter Two

Chapter 2
Background

In this chapter we will show the basic theoretical information about some technologies to be used in our project.

2.1ZigBee and other competitors

Wireless sensor networks are one of the most rapidly growing technologies and has a wide variety of applications. As a wireless technology, ZigBee becomes popular in the recent years due to its ultra-low power consumption. However, there are many wireless technologies that are competing with the ZigBee such as Wi-Fi and Bluetooth. Wi-Fi is based on the IEEE 802.11 standards and considered as the most prevalent technology in the world. Most researchers prefer to exploit Wi-Fi technology in their projects due its availability in almost everywhere, and in every wireless device (such as laptops, smart phones, tablets and so on). A typical wireless router uses 802.11b or 802.11g with a stock antenna might have a communication range of 32 m indoors and 95 m outdoors. Wi-Fi transceivers are very suitable for applications that requires high data rate and/or an internet connection.

On the other hand, Bluetooth is primarily designed for low power consumption and short ranges communication (1m and 10m). Bluetooth is not targeted for a specific application, but supports a multitude of applications. Therefore the technology has been adopted by wide variety of devices, including computers, cell phones, headsets, PDAs and cars.

2.1.1ZigBee Physical Layer (802.15.4)

ZigBee is a wireless technology developed as an open global standard to address the unique needs of low-cost, low-power wireless M2M networks. The ZigBee standard operates on the IEEE 802.15.4 physical radio specification and operates in unlicensed bands including 2.4 GHz, 900 MHz and 868 MHz

The 802.15.4 specification upon which the ZigBee stack operates gained ratification by the Institute of Electrical and Electronics Engineers (IEEE) in 2003. The specification is a packet-based radio protocol intended for low-cost, battery-operated devices. The protocol allows devices to communicate in a variety of network topologies and can have battery life lasting several years.

ZigBee protocol has been created and ratified by member companies of the ZigBee Alliance. Over 300 leading semiconductor manufacturers, technology firms, OEMs and service companies comprise the ZigBee Alliance membership. The ZigBee protocol was designed to provide an easy-to-use wireless data solution characterized by secure, reliable wireless network architectures.

2.1.2ZigBee Advantage

The ZigBee protocol is designed to communicate data through hostile RF environments that are common in commercial and industrial applications.

ZigBee protocol features include:

- Support for multiple network topologies such as point-to-point, point-to-multipoint and mesh networks.
- Low duty cycle – provides long battery life.
- Low latency.
- Direct Sequence Spread Spectrum (DSSS).
- Up to 65,000 nodes per network.
- 128-bit AES encryption for secure data connections.
- Collision avoidance, retries and acknowledgements.

2.1.3Mesh Networks

A key component of the ZigBee protocol is the ability to support mesh networking. In a mesh network, nodes are interconnected with other nodes so that multiple pathways connect each node. Connections between nodes are dynamically updated and optimized through sophisticated, built-in mesh routing table.

Mesh networks are decentralized in nature; each node is capable of self-discovery on the network. Also, as nodes leave the network, the mesh topology allows the nodes to reconfigure routing paths based on the new network structure. The characteristics of mesh topology and ad-hoc routing provide greater stability in changing conditions or failure at single nodes.

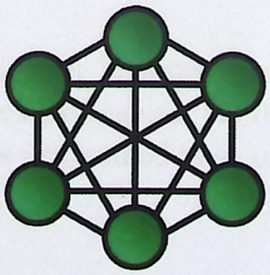


Figure 2.1:Mesh Networks.

2.1.4 Comparison between Wi-Fi, Bluetooth and ZigBee technologies

	Bluetooth	ZigBee	Wi-Fi
Frequency Band	2.4 GHz	868/915 MHz, 2.4 GHz	2.4 GHz, 5 GHz
Data rate	1 Mbps	250 Kbps	54 Mbps
Nominal range	10 m	10 -100 m	50 -100 m
# RF channels	79	16 (for 2.4 GHz)	14 (for 2.4 GHz)
Channel bandwidth	1 MHz	2 MHz (for 2.4 GHz)	22 MHz
Power Consumption	Medium	Low	High

Table 2.1: Comparison between technologies.[3]

We chose ZigBee because all the benefits mentioned previously, and the ZigBee nodes can go from sleep mode to active mode in 30 ms or less, that save power.

2.1.5I/O Data Format

I/O data begins with a header. The first byte of the header defines the number of samples forthcoming. The last 2 bytes of the header (Channel Indicator) define which inputs are active. Each bit represents either a DIO line or ADC channel.

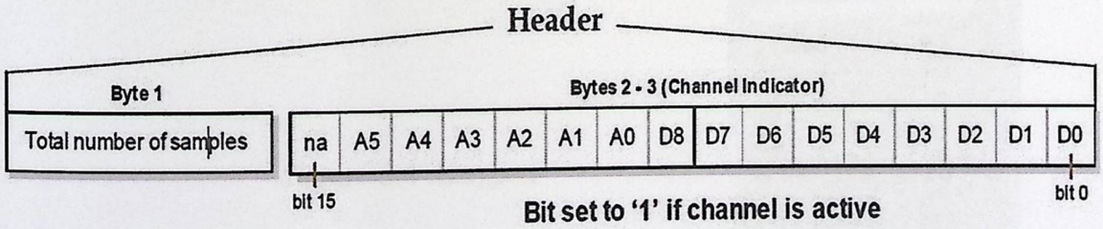


Figure 2.2 : Data Format.

Header Sample data follows the header and the channel indicator frame is used to determine how to read the sample data. If any of the DIO lines are enabled, the first 2 bytes are the DIO sample. The ADC data follows. ADC channel data is represented as an unsigned 10-bit value right-justified on a 16- bit boundary.

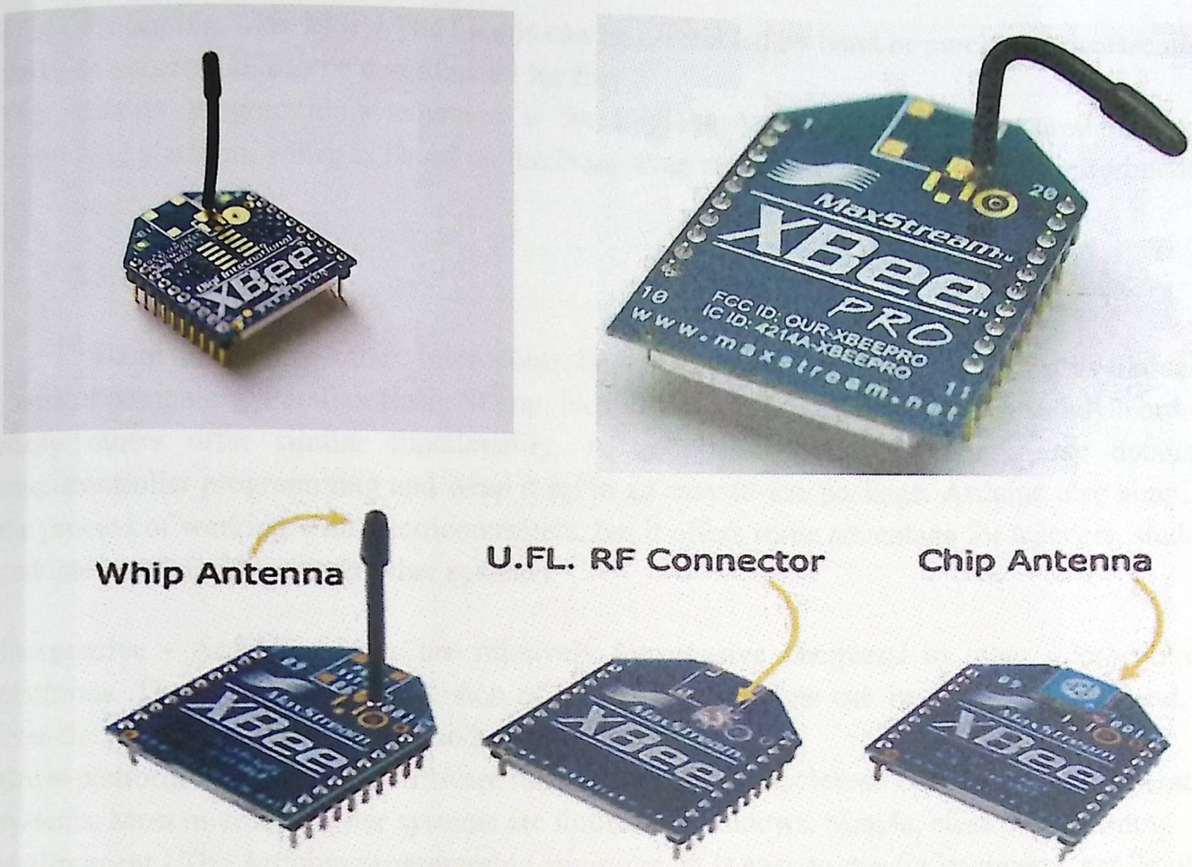


Figure 2.3 : ZigBee Examples

2.2 Keypad

This unit use to insert a correct password to open the park, that connect to Arduino to control the Keypad.

Figure 2.4 :Keypad

2.3Arduino [4]

Arduino is a tool for making computers that can sense and control more of the physical world than your desktop computer. It's an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board. Arduino can be used to develop interactive objects, taking inputs from a variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs. Arduino projects can be stand-alone, or they can communicate with software running on your computer (e.g.

Flash, Processing, Max MSP.) The boards can be assembled by hand or purchased preassembled; the open-source IDE can be downloaded for free.
The Arduino programming language is an implementation of Wiring, a similar physical computing platform, which is based on the Processing multimedia programming environment.

2.3.1Why Arduino?

There are many other microcontrollers and microcontroller platforms available for physical computing. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, and many others offer similar functionality. All of these tools take the messy details of microcontroller programming and wrap it up in an easy-to-use package. Arduino also simplifies the process of working with microcontrollers, but it offers some advantage for teachers, students, and interested amateurs over other systems:

- Inexpensive - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than $50 .
- Cross-platform - The Arduino software runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows. Simple, clear programming environment - The Arduino programming environment is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with the look and feel of Arduino.

Open source and extensible software- The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.

Open source and extensible hardware - The Arduino is based on Atmel's ATmega8 and ATmega168 microcontrollers. The plans for the modules are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

2.3.2Types of Arduino

1- Arduino Uno:

The Arduino Uno is a microcontroller board based on the ATmega328 .It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic

resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.

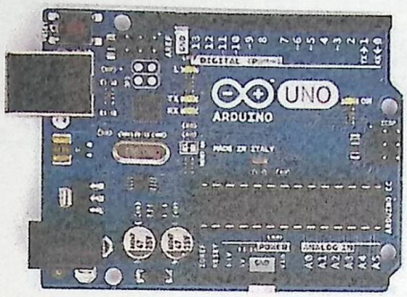


Figure 2.5 :Arduino Uno.

2- Arduino GSM Shield : [5]

The Arduino GSM Shield connects your Arduino to the internet using the GPRS wireless network. Just plug this module onto your Arduino board, plug in a SIM card from an operator offering GPRS coverage and follow a few simple instructions to start controlling your world through the internet. You can also make/receive voice calls (you will need an external speaker and microphone circuit) and send/receive SMS messages.

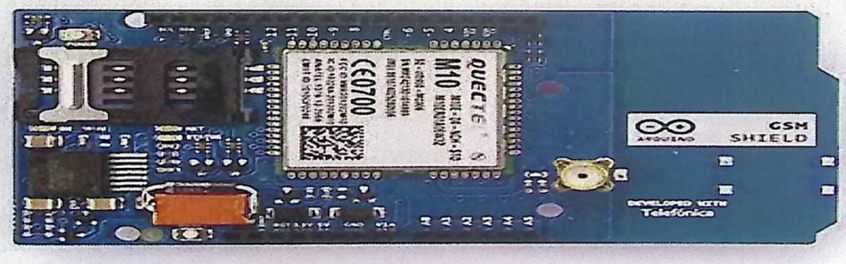


Figure 2.6 :Arduino GSM Shield .

3- Arduino MEGA ADK :

The Arduino MEGA ADK is a microcontroller board based on the ATmega2560 . It has a USB host interface to connect with Android based phones, based on the MAX3421e IC. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button.

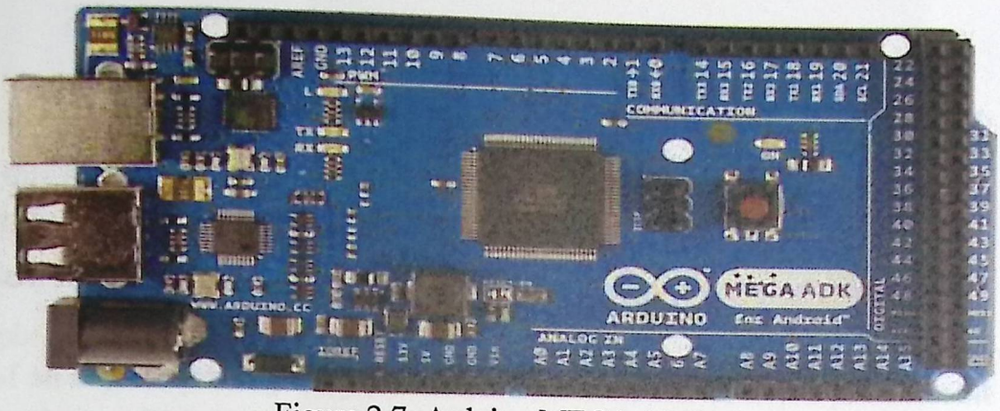


Figure 2.7 :Arduino MEGA ADK .

2.4 Global System for Mobile

GSM (Global System for Mobile communication) is a digital mobile telephony system that is widely used in Europe and other parts of the world. GSM uses a variation of time division multiple access (TDMA) and is the most widely used of the three digital wireless telephony technologies.

Mobile services based on GSM technology were first launched in Finland in 1991. Today, more than 690 mobile networks provide GSM services across 213 countries and GSM represents 82.4% of all global mobile connections. According to GSM World, there are now more than 2 billion GSM mobile phone users worldwide. GSM World references China as "the largest single GSM market, with more than 370 million users, followed by Russia with 145 million, India with 83 million and the USA with 78 million users."

Since many GSM network operators have roaming agreements with foreign operators, users can often continue to use their mobile phones when they travel to other countries. SIM cards (Subscriber Identity Module) holding home network access configurations may be switched to those will metered local access, significantly reducing roaming costs while experiencing no reductions in service.

GSM, together with other technologies, is part of the evolution of wireless mobile telecommunications that includes High-Speed Circuit-Switched Data (HSCSD), General Packet Radio System (GPRS), Enhanced Data GSM Environment (EDGE), and Universal Mobile Telecommunications Service (UMTS).

2.5 Sensors

In this section will discussion types of sensors may will use it in our project, and will Comparison between this types.

2.5.1Photoelectric sensor

A photoelectric sensor, or photo eye, is a device used to detect the distance, absence, or presence of an object by using a light transmitter, often infrared, and a photoelectric receiver. They are used extensively in industrial manufacturing. There are three different functional types: opposed (through beam), retro-reflective, and proximity-sensing (diffused).

A through beam arrangement consists of a receiver located within the line-of-sight of the transmitter. In this mode, an object is detected when the light beam is blocked from getting to the receiver from the transmitter.

A retro reflective arrangement places the transmitter and receiver at the same location and uses a reflector to bounce the light beam back from the transmitter to the receiver. An object is sensed when the beam is interrupted and fails to reach the receiver.

A proximity-sensing (diffused) arrangement is one in which the transmitted radiation must reflect off the object in order to reach the receiver. In this mode, an object is detected when the receiver sees the transmitted source rather than when it fails to see it.

Some photo eyes have two different operational types, light operate and dark operate. Light operate photo eyes become operational when the receiver "receives" the transmitter signal. Dark operate photo eyes become operational when the receiver "does not receive" the transmitter signal.

The detecting range of a photoelectric sensor is its "field of view", or the maximum distance from which the sensor can retrieve information, minus the minimum distance. A minimum detectable object is the smallest object the sensor can detect. More accurate sensors can often have minimum detectable objects of minuscule size.

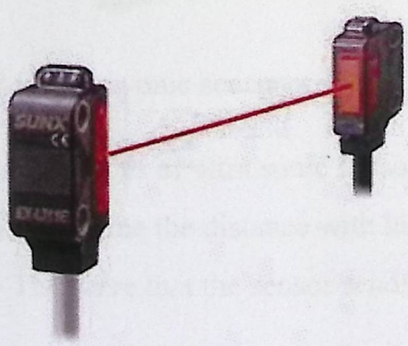


Figure 2.8 : Photoelectric sensor.

2.5.2Ultrasonic sensor

Ultrasonic sensor is very good type of sensors, this sensor is used as a transceiver (transmitter and receiver), and the working system of this sensor like the radar, which evaluate attributes of a target by interpreting the echoes from radio or sound waves. Ultrasonic makes high frequency sound wave and evaluates the echo which received back by the sensors and measure the time between sending and receiving the signal to determine the distance. This sensor can use in a lot of applications like measuring wind speed.

Ultra sonic used transducer to convert sound waves to above sound waves and the range of the frequency above 18KHz.This process happened from convert electrical signal to the sound and when receive this signal it converts the signal from sound to electrical and we can measure it.

We choose using ultrasonic sensor than photoelectric sensor, because it is more accurate and reliable, and both of them have the same cost.

From our studying a lot of light sensors we find that we can circumvent on them, and they will never give the result we need.

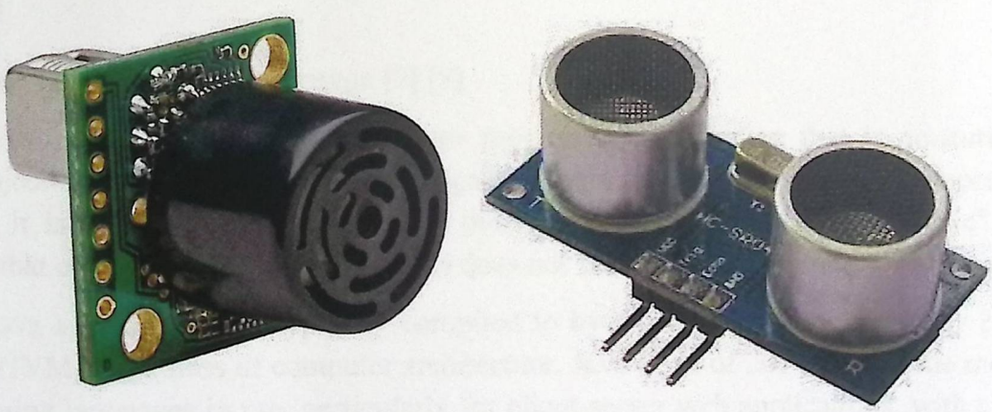


Figure 2.9 : Ultrasonic sensor.

- ✓ Advantages of ultra sonic sensor :

1- Determine the distance with high accuracy..

2- The wave that the sensor sends it is does not affect with human.

- ✓ Disadvantages of photoelectric sensor :

1- This sensor determines that these is an object with quite accuracy.

2- We can cut the light using any object.

2.6 Software

The computer can't work independently, it need for the software to manage, read, write, analysis, and save data and do other operations.

There are a lot of programming languages that are used to build a computer program, choosing any language depends on the tasks of the program, we need any language can handling with database.

2.6.1C# Programming Language [6]

C# (pronounced as see sharp) is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed by Microsoft within its .NET initiative and later approved as a standard by Ecma (ECMA-334) and ISO (ISO/IEC 23270:2006). C# is one of the programming languages designed for the Common Language Infrastructure.

A multi-paradigm programming language is a programming language that supports more than one programming paradigm. As Leda designer Timothy Budd puts it: "The idea of a multi paradigm language is to provide a framework in which programmers can work in a variety of styles, freely intermixing constructs from different paradigms. The design goal of such languages is to allow programmers to use the best tool for a job, admitting that no one paradigm solves all problems in the easiest or most efficient way.

2.6.2 Java Programming Language [7] [8]

Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another.

Java applications are typically compiled to byte code that can run on any Java virtual machine (JVM) regardless of computer architecture. Java is, as of 2014, one of the most popular programming languages in use, particularly for client-server web applications, with a reported 9 million developers.

Java was originally developed by James Gosling at Sun Microsystems (which has since merged into Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them. [9]

There are five primary goals in the Java language:

- It must be simple, object-oriented and familiar.
- It must be robust and secure.

- It must be architecture-neutral and portable.
- It must execute with high performance.
- It must be interpreted, threaded, and dynamic.

Figure 2.10: C# Programming Language Logo

Figure 2.11: Java Programming Language Logo

2.6.3Microsoft Office Access [10]

Microsoft Access, also known as Microsoft Office Access, is a database management system from Microsoft that combines the relational Microsoft Jet Database Engine with a graphical user interface and software-development tools. It is a member of the Microsoft Office suite of applications, included in the Professional and higher editions or sold separately.

Microsoft Access stores data in its own format based on the Access Jet Database Engine. It can also import or link directly to data stored in other applications and databases.

Software developers and data architects can use Microsoft Access to develop application software, and "power users" can use it to build software applications. Like other Office applications, Access is supported by Visual Basic for Applications (VBA), an object-oriented programming language that can reference a variety of objects including DAO (Data Access Objects), ActiveX Data Objects, and many other ActiveX components. Visual objects used in forms and reports expose their methods and properties in the VBA programming environment, and VBA code modules may declare and call Windows operating-system functions.

2.6.4 SQLite Database[11]

SQLite is an open source database. SQLite supports standard relational database features like SQL syntax, transactions, and prepared statements. The database requires limited memory at run time (approx. 250 Kbytes) which makes it a good candidate from being embedded into other run times.

SQLite is embedded into every Android device. Using an SQLite database in Android does not require a setup procedure or administration of the database.

You only have to define the SQL statements for creating and updating the database. Afterwards the database is automatically managed for you by the Android platform. Access to an SQLite database involves accessing the file system. This can be slow. Therefore it is recommended to perform database operations asynchronously.

2.6.5Arduino Software

The Arduino software is provided to you "as is" and we make no express or implied warranties whatsoever with respect to its functionality, operability, or use, including, without limitation, any implied warranties of merchantability, fitness for a particular purpose, or infringement. we expressly disclaim any liability whatsoever for any direct, indirect, consequential, incidental or special damages, including, without limitation, lost revenues, lost profits, losses resulting from business interruption or loss of data, regardless of the form of action or legal theory under which the liability may be asserted, even if advised of the possibility or likelihood of such damages.

The open-source Arduino environment makes it easy to write code and upload it to the i/o board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing, avr-gcc, and other open source software.

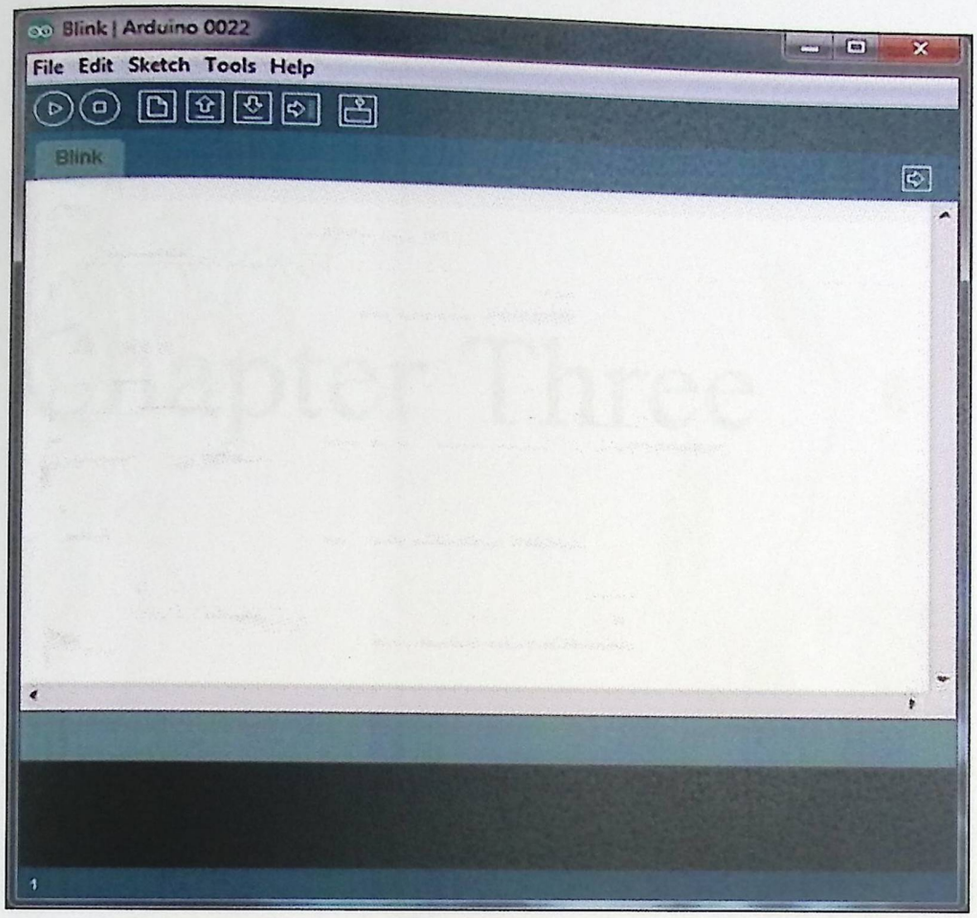


Figure 2.12: Arduino Software.

Chapter Three

Chapter 3
System Design

This chapter explain the design concepts, general block diagram of the system, the system how will work, and system data flow.

3.1The General Block Diagram

The General Block Diagram consist the main components of our project that as shown in figure 3.1.

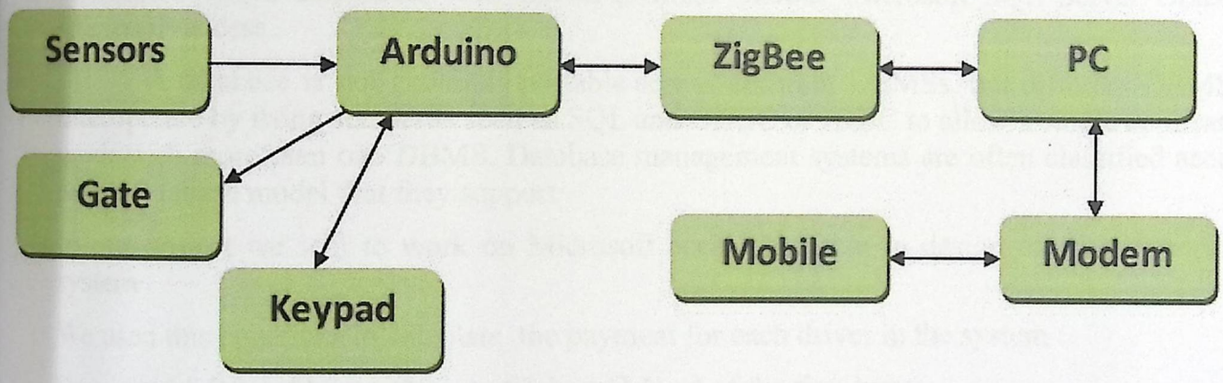


Figure 3.1: Block Diagram For The System .

3.2Billing

We will use three method for charging:

1. Using mobile balance for the driver (that need to deal with communication company).

   That mean to taking the value that needed from the mobile balance, and if the balance was not enough will be agreed with the mobile company to allow debt to specific value

2. Using a charging card that is special for the parking that use this APS system to allow the driver to use the card more than one until the balance of the card is finish .

3. The traditional method to pay to the employee and take the password for the gate . (this method is used if no prior reservation have been made) .

3.3Data Base

A database is an organized collection of data. The data is typically organized to model aspects of reality in a way that supports processes requiring information.

Database management systems (DBMSs) are computer software applications that interact with the user, other applications, and the database itself to capture and analyze data. A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases. Well-known DBMSs include Microsoft SQL Server, Oracle and Microsoft access .

A database is not generally portable across different DBMSs, but different DBMSs can interoperate by using standards such as SQL and ODBC or JDBC to allow a single application to work with more than one DBMS. Database management systems are often classified according to the database model that they support .

In our project we will to work on Microsoft access program to design the data base for this system .

We used this equations to calculate the payment for each driver in the system :

Payment =[(No. of hours)*(cost of 1-hour)]-(cost of the first hour)

We assume the cost of one hour = 1.0 $

NO	Mobile Number	Parking slot Number	Gate password	Parking time	Time Exit	Payment
1	025896345	4	0000000	11:00	01:00	1$
2						
. . .						

Table3.1: The data base designed for the system.

3.4Main System Components

This section will describe the role of each component :

3.4.1 Sensors

Ultrasonic sensors transmit ultrasonic waves from its sensor head and again receives the ultrasonic waves reflected from an object, by measuring the length of time from the transmission to reception of the sonic wave, it detects the position of the object.

The sound velocity in the atmosphere varies with the temperature of the atmosphere. The sound velocity is 331.45 m/s in the atmosphere at 0°C.

In our project we decided to use ultrasonic sensor since it is specifications meet the requirement of our project. We decide to use Ultrasonic Ranging Detector Mod HC-SR04 Distance Sensor with the next specifications:

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:

Using IO trigger for at least 10us high level signal, The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.

IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.

Its stable performance and high ranging accuracy make it a good choice for us. Compared to the Shape IR ranging module, HC-SR04 is more inexpensive than it. But it has the same ranging accuracy and longer ranging distance.

There are 4 pins out of the module: VCC , Trig, Echo, GND . So it's a very easy interface for controller to use it ranging. The all process is : pull the Trig pin to high level for more than 10us impulse , the module start ranging ; finish ranging , If you find an object in front , Echo pin will be high level , and based on the different distance, it will take the different duration of high level. So we can calculated the distance easily :

Distance = ((Duration of high level)*(Sonic :340m/s))/2 …… Equations (2.1)

3.4.1.1 Principle of operation

At the beginning we should get the testing values (distances) that will be used to compare the reading of the sensor with The testing value will be get from sending ultrasonic signal from the sensor to an object putted at the maximum distance between the ground and the car bottom (50cm as we suppose).

The reflection of the wave will cause a signal with specific duration on the echo pin entered to the Arduino. Using a code in the Arduino we will store this value as a reference value.

In the system, the decision if there is a car in the space of parking or not depends on the transmitted ultrasonic waves from its sensor head and again receives the ultrasonic waves reflected from an object. The reflected wave should hit the sensor in a time equals to two the travel time of the ultrasonic signal from the sensor to the car bottom. The signal with specific

duration will be taken from the echo pin in the sensor and manipulated in the Arduino. After that it will be compared to a testing distances stored before and then get the decision if there is car in that space or not.

3.4.2Arduino Mega

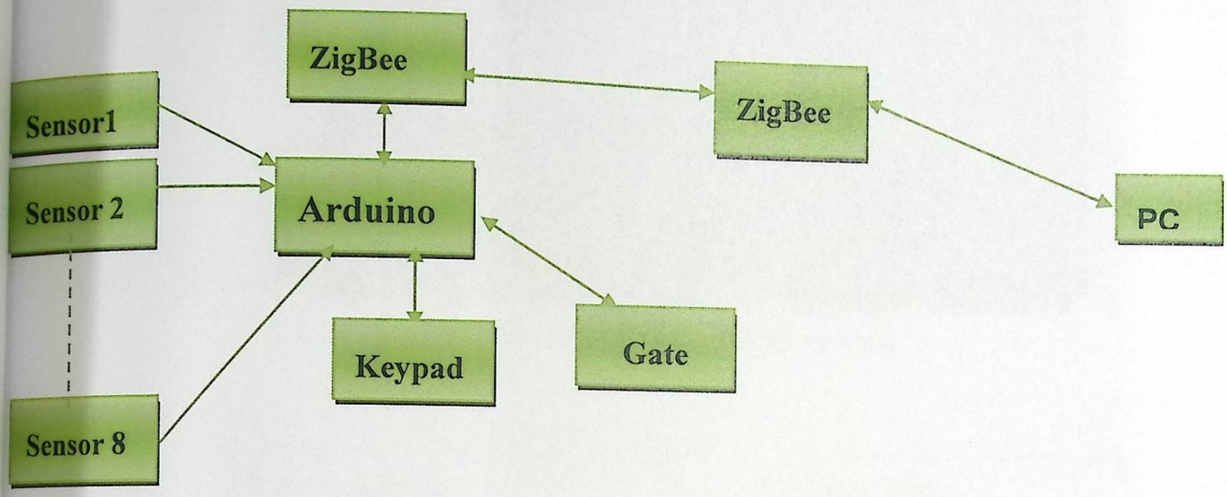


Figure 3.2: The block diagram of the Arduino work in the system .

the Arduino read the status of sensors then it will generate an 8-bit string of zeros and ones , zeros are empty and ones are reserved due to the gate situation , then it will send this string over ZigBee ,until it reaches the central Arduino which will pass these information to the PC.

After the reservation complete , PC will send the gate number and the code to central Arduino , which will direct them to the Arduino connected to the gate , after the code and reservation information received by the Arduino will activate the relay at the gate which come including data.

When The car arrive, the driver must enter the code which get it into the keypad connected to the Arduino, which will compare the code entered with the received one from the PC, if there were a match then the gate will open to the car in.

When the car in , the sensor will changes its status ,sends its new status to the Arduino and then returned the process again.

The Arduino mega is a microcontroller based on the Atmega1280 .it has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack,

an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Mega is compatible with most shields designed for the Arduino Duemilanove .

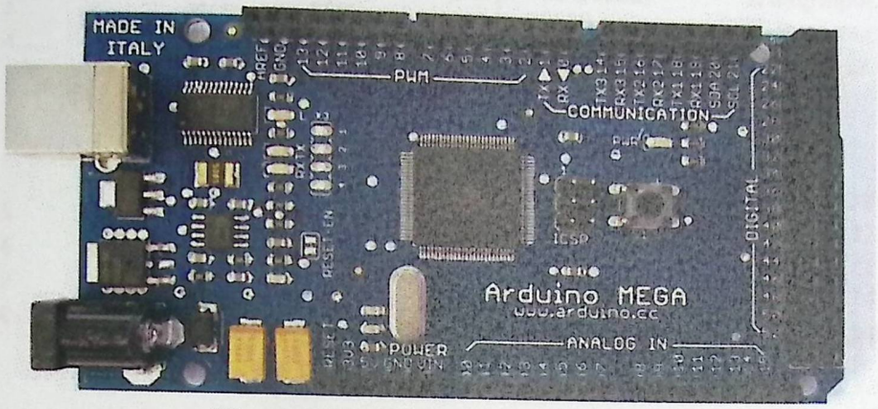


Figure 3.3: Arduino Mega module .

Microcontroller	AT mega 1280
Operating Voltage	5 V
Input Voltage (recommended)	7-12 V
Input Voltage (limits)	6-20 V
Digital I\O pins	54
Analog input pins	16
DC current per i\o pins	40mA
DC current for 3.3 V pin	50mA
Flash Memory	128 KB
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

Table 3.2: Characteristics about Arduino Mega .

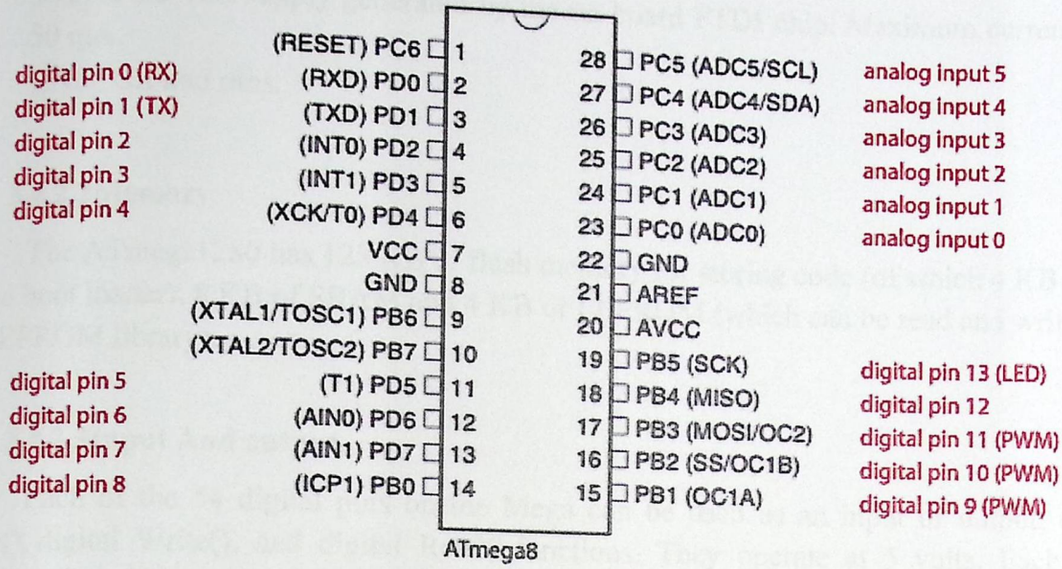


Figure 3.4: The data sheet for Arduino Mega .

3.4.2.1Power

The Arduino Mega can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the power connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- Vin :- The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

- 5V. The regulated power supply used to power the microcontroller and other components on the board. This can come either from Vin via an on-board regulator, or be supplied by USB or another regulated 5V supply.

- 3V3. A 3.3 volt supply generated by the on-board FTDI chip. Maximum current draw is 50 mA.
- GND. Ground pins.

3.4.2.2Memory

The ATmega1280 has 128 KB of flash memory for storing code (of which 4 KB is used for the boot loader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

3.4.2.3Input And output

Each of the 54 digital pins on the Mega can be used as an input or output, using pin Mode(), digital Write(), and digital Read() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX). Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the FTDI USB-to-TTL Serial chip.
- External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2). These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attach Interrupt()function for details.
- PWM: 2 to 13 and 44 to 46. Provide 8-bit PWM output with the analog Write() function.
- SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Duemilanove and Diecimila .

The Mega has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analog Reference() function.

- AREF. Reference voltage for the analog inputs. Used with analog Reference().
- Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

3.4.2.4Communication

The Arduino Mega has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega1280 provides four hardware UARTs for TTL (5V) serial communication. An FTDI FT232RLon the board channels one of these over USB and the FTDI drivers (included with the Arduino software) provide a virtual com port to software on the computer. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board .

The RX and TX LEDs on the board will flash when data is being transmitted via the FTDI chip and USB connection to the computer (but not for serial communication on pins 0 and (1), A Software Serial library allows for serial communication on any of the Mega's digital pins.

The ATmega1280 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus.

Physical characteristic and shield compatibility :-

The maximum length and width of the Mega PCB are 4 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

The Mega is designed to be compatible with most shields designed for the Diecimila or Duemilanove. Digital pins 0 to 13 (and the adjacent AREF and GND pins), analog inputs 0 to 5, the power header, and ICSP header are all in equivalent locations. Further the main UART (serial port) is located on the same pins (0 and 1) (pins 2 and 3 respectively).

3.4.2.5 Interfacing between Arduino and other device

This board explain interface between Arduino and Ultrasonic sensor, how the pin of sensor connect with pin of Arduino.

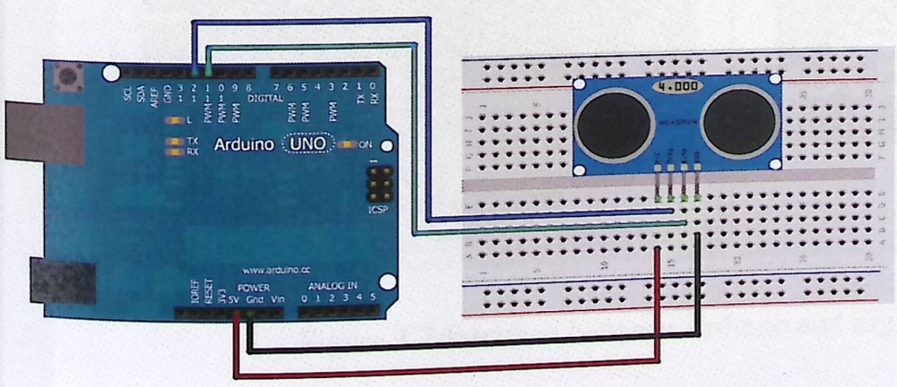


Figure 3.5:Interface between Arduino and Ultrasonic sensor.

Figure 3.6 explain interface between Arduino, and Ultrasonic sensor, and slots of park, this circuit is a prototype for the subsystem of Arduino.

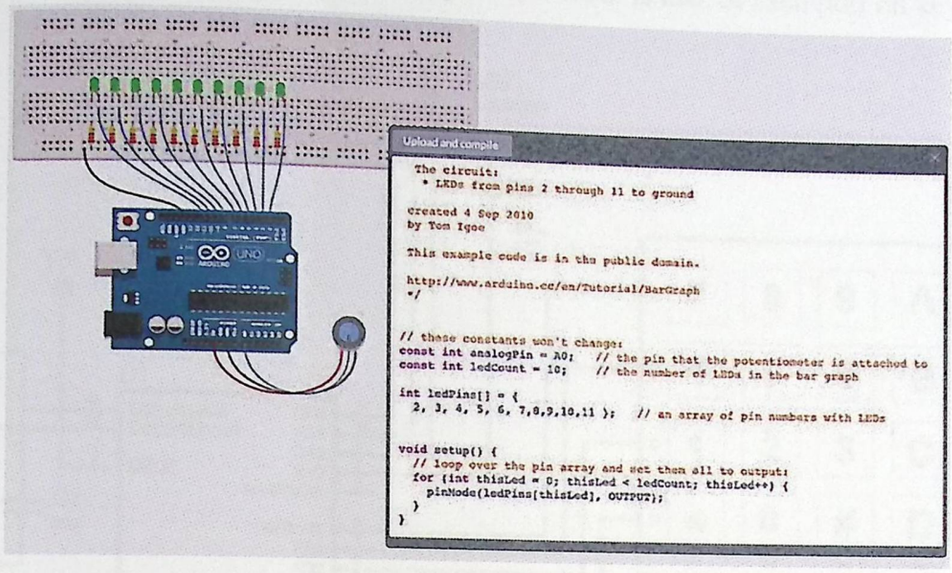


Figure 3.6: A prototype for the subsystem of Arduino.

Figure 3.7explain interface between Arduino and ZigBee, how the pin of ZigBee connect with pin of Arduino , ZigBee will work as transmission and received device.

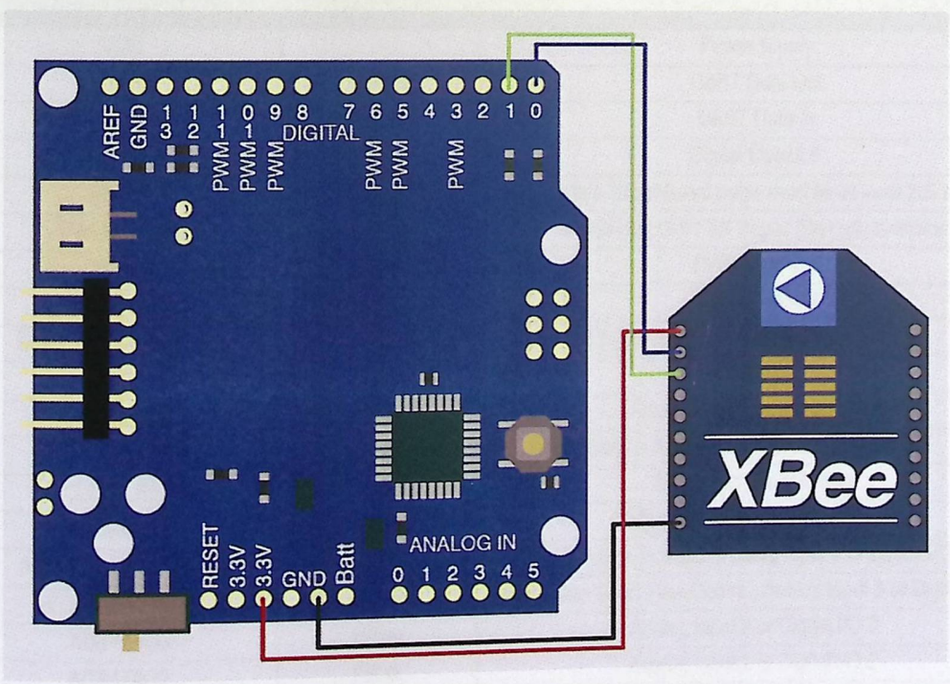


Figure 3.7:Interface between Arduino and ZigBee.

3.4.3Keypad

This is the internal component of keypad, and what is task of each port on keypad.

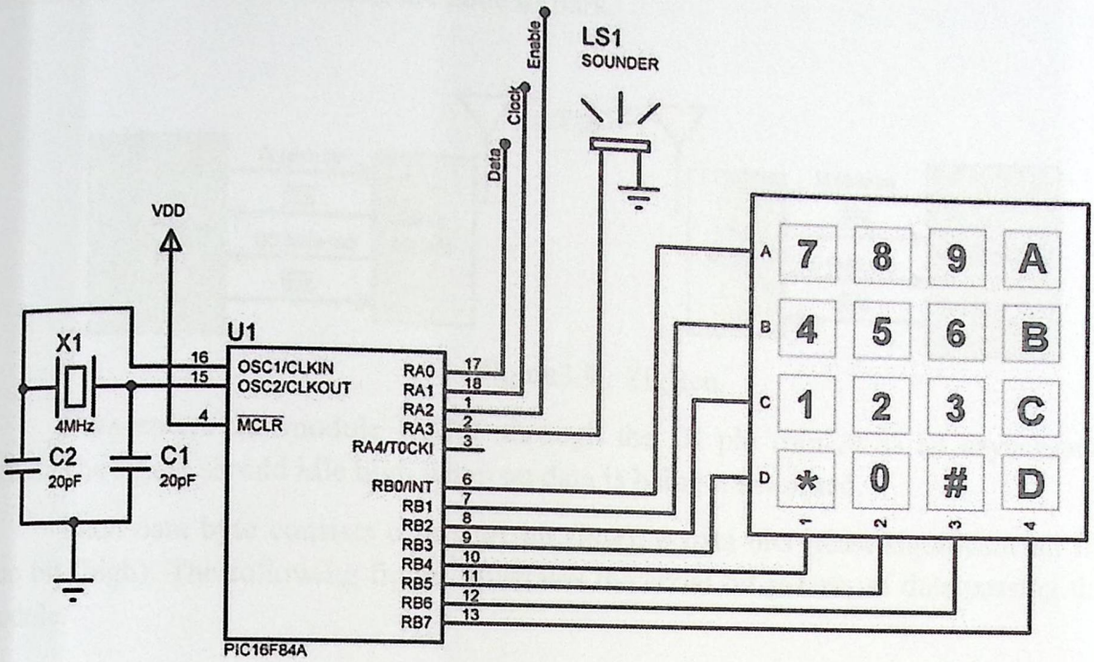


Figure3.8 : internal component of keypad.

3.4.4ZigBee

Pin #	Name	Direction	Description
1	VCC	-	Power supply
2	DOUT	Output	UART Data Out
3	DIN / $\overline{\text{CONFIG}}$	Input	UART Data In
4	$\overline{\text{DO8}}$*	Output	Digital Output 8
5	$\overline{\text{RESET}}$	Input	Module Reset (reset pulse must be at least 200 ns)
6	PWM0 / RSSI	Output	PWM Output 0 / RX Signal Strength Indicator
7	PWM1	Output	PWM Output 1
8	[reserved]	-	Do not connect
9	$\overline{\text{DTR}}$ / SLEEP_RQ / DI8	Input	Pin Sleep Control Line or Digital Input 8
10	GND	-	Ground
11	AD4 / DIO4	Either	Analog Input 4 or Digital I/O 4
12	$\overline{\text{CTS}}$ / DIO7	Either	Clear-to-Send Flow Control or Digital I/O 7
13	ON / $\overline{\text{SLEEP}}$	Output	Module Status Indicator
14	VREF	Input	Voltage Reference for A/D Inputs
15	Associate / AD5 / DIO5	Either	Associated Indicator, Analog Input 5 or Digital I/O 5
16	$\overline{\text{RTS}}$ / AD6 / DIO6	Either	Request-to-Send Flow Control, Analog Input 6 or Digital I/O 6
17	AD3 / DIO3	Either	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Either	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Either	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0	Either	Analog Input 0 or Digital I/O 0

Table 3.3 : Datasheet for ZigBee.

This component used to sending and received the data from and to PC, and from and to Arduino, and used to addressing the zone of park.

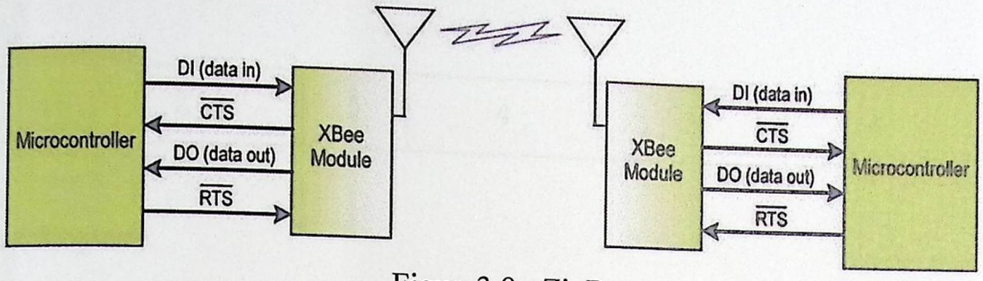


Figure3.9 : ZigBee.

Data enters the module UART through the DI pin (pin 3) as an asynchronous serial signal. The signal should idle high when no data is being transmitted.

Each data byte consists of a start bit (low), 8 data bits (least significant bit first) and a stop bit (high). The following figure illustrates the serial bit pattern of data passing through the module.

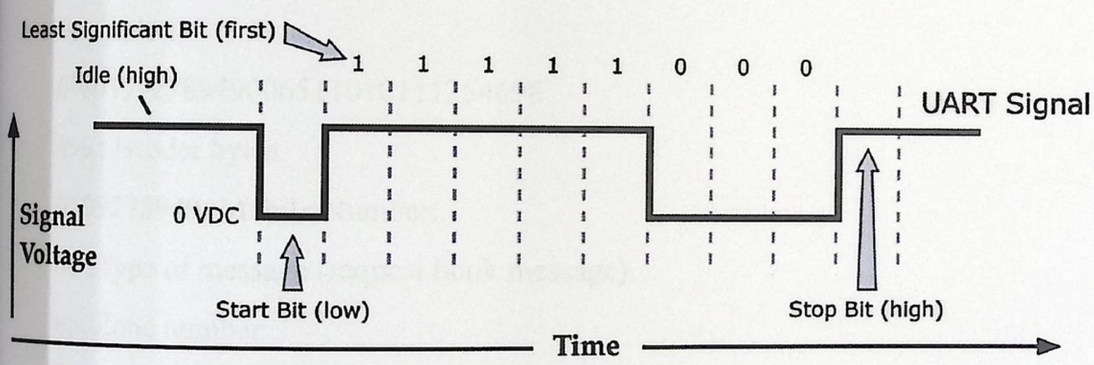


Figure3.10 : data packet as transmitted through the ZigBee.

3.4.4.1 Packet format analysis

The packet format which is going to be transferred between our subsystems, in our project could be three type of message: request book message, confirmation message, and cancellation book message.

1	2	3	4	5	6

1: Header (2 bytes).

2: Mobile Number (5 bytes).

3: Type of message (2 bits).

4: Zone number (1 byte).

5: Status of sensors (1 byte).

6: Password to open the slot (3 byte).

Example 1 :

#%0599278949006511010111254698

#% : Header bytes.

0599278949 :Mobile Number.

00 :Type of message (request book message).

65: Zone number.

11010111 : Status of sensors, 1 means full, 0 means empty.

254698: Password to open the slot.

Example 2 :

##0599123456118801110100569875

: Header bytes.

0599123456 : Mobile Number.

11 : Type of message (cancellation book message).

88 : Zone number.

01110100: Status of sensors, 1 means full, 0 means empty.

569875: Password to open the slot.

Example 3 :

&@0598987456012501110100457425

&@: Header bytes.

0598987456 : Mobile Number.

01 : Type of message (confirmation message).
25 : Zone number.

01110100 **:** Status of sensors, 1 means full, 0 means empty.

457425: Password to open the slot.

3.4.4.2Addressing

Every RF data packet sent over-the-air contains a Source Address and Destination Address field in its header. The RF module conforms to the 802.15.4 specification and supports both short 16-bit addresses and long 64-bit addresses. A unique 64-bit IEEE source address is assigned at the factory and can be read with the SL (Serial Number Low) and SH (Serial Number High) commands. Short addressing must be configured manually. A module will use its unique 64-bit address as its Source Address if it's MY (16-bit Source Address) value is "0xFFFF" or "0xFFFE".

To send a packet to a specific module using 64-bit addressing: Set the Destination Address (DL + DH) of the sender to match the Source Address (SL + SH) of the intended destination module.

To send a packet to a specific module using 16-bit addressing: Set DL (Destination Address Low) parameter to equal the MY parameter of the intended destination module and set the DH (Destination Address High) parameter to '0'.

We use addressing to divide the park to zones, and give each zone Special address that used to reach it easily, and that be the park more organized.

Chapter Four

Chapter 4
Software Requirement Specification

In this chapter, we describe the requirements of our project, making simple scenarios to show how the project works.

4.1 Requirement Description

4.1.1 Actors

4.1.1.1 Administrator

1. Logging on and access the system.
2. Maintain the system.
3. Monitor the system.

4.1.1.2 Driver

1. Request reservation.
2. Enter or exit car in certain slot.

4.1.2 Usage Scenario

4.1.2.1 Administrator

4.1.2.1.1 Logging on and access the system

Use Case	Logging on and access the system
Primary Actor	Administrator
Goal in Context	Enable the admin to access the system.
Scenario	1. The admin log on system. 2. The admin chooses display data on screen. 3. The system preview a list's of data. 4. The admin can deal with data (delete, add, and update the data).

Table 4.1 : Logging on and access the system.

4.1.2.1.2 Maintain the system.

Use Case	Maintain the system
Primary Actor	Administrator
Goal in Context	The admin able to Maintain the system.
Scenario	1. The admin log on system. 2. The admin checks display data on screen. 3. The admin checks GSM transceiver andZigBee transceiver performance by send and receive messages.

Table 4.2 : Maintain the system.

4.1.2.1.3Monitor the system.

Use Case	Monitor the system
Primary Actor	Administrator
Goal in Context	The admin able to Monitor the system.
Scenario	1. The admin log on system. 2. The admin can display and see the data flow to and from server. 3. The admin checks GSM transceiver and ZigBee transceiver performance by send and receive messages . 4. The admin monitor the park and mange it.

Table 4.3 : Monitor the system.

4.1.2.2Driver

4.1.2.2.1 Request Reservation

Use Case	Request Reservation
Primary Actor	Driver
Goal in Context	The driver can request reservation remotely.
Scenario	1. The driver send SMS to GSM park for request reservation. 2. The system will ask driver to confirm request. 3. The driver send SMS to GSM park for confirm request. 4. The system will reserve slot , then send to driver details of reserved slot.

Table 4.4 : Request Reservation.

4.1.2.2.2Enter or exit car in certain slot

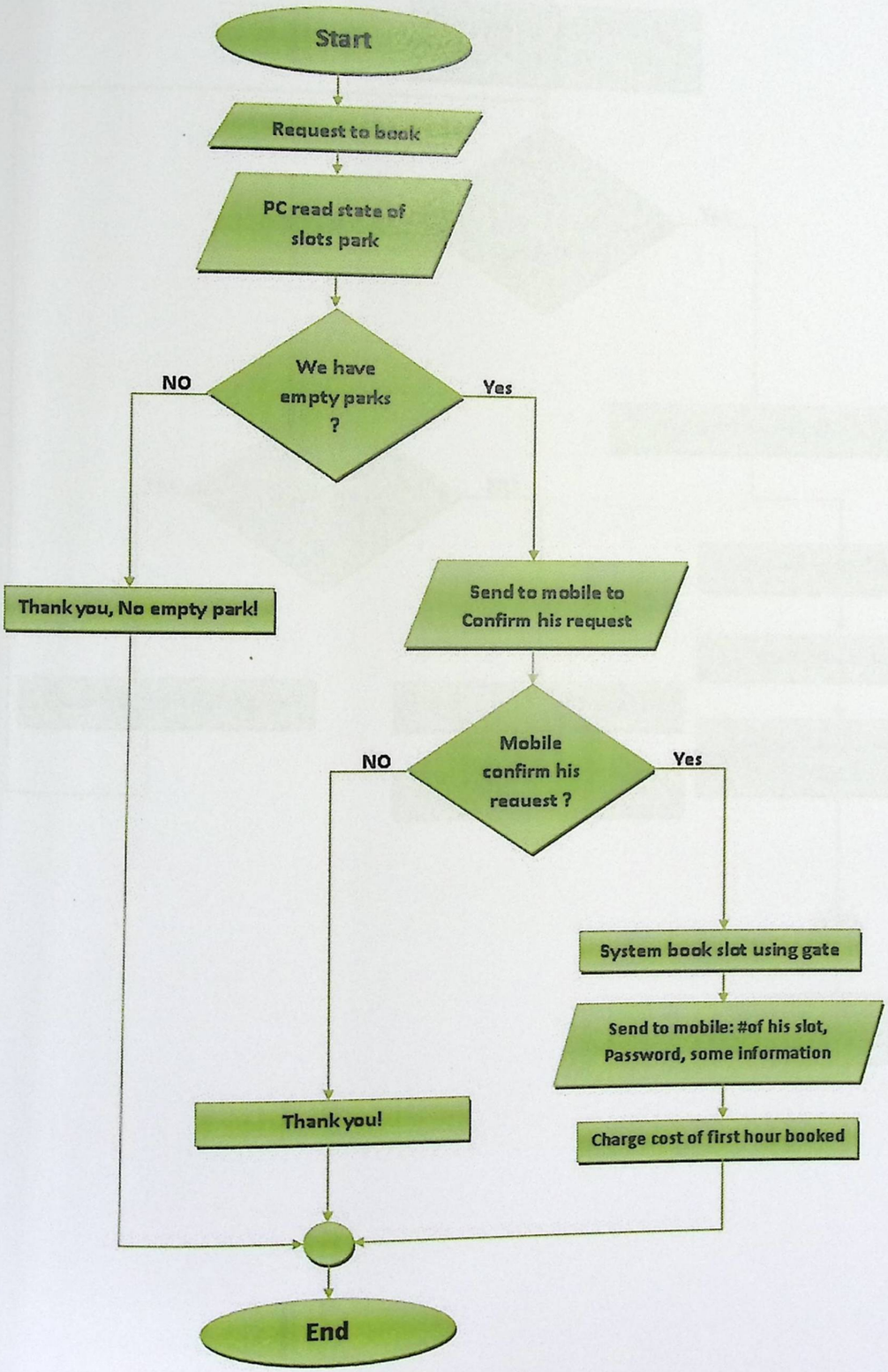
Use Case	Enter or exit car in certain slot
Primary Actor	Driver
Goal in Context	The driver can Enter or exit car in certain slot that reserved remotely.
Scenario	1. The driver enter the password to open slot. 2. The system will check the password, if it's correct the slot will open . 3. The driver Enter or exit car in certain slot that reserved remotely.

Table 4.5 : Enter or exit car in certain slot.

4.2 Flow Chart

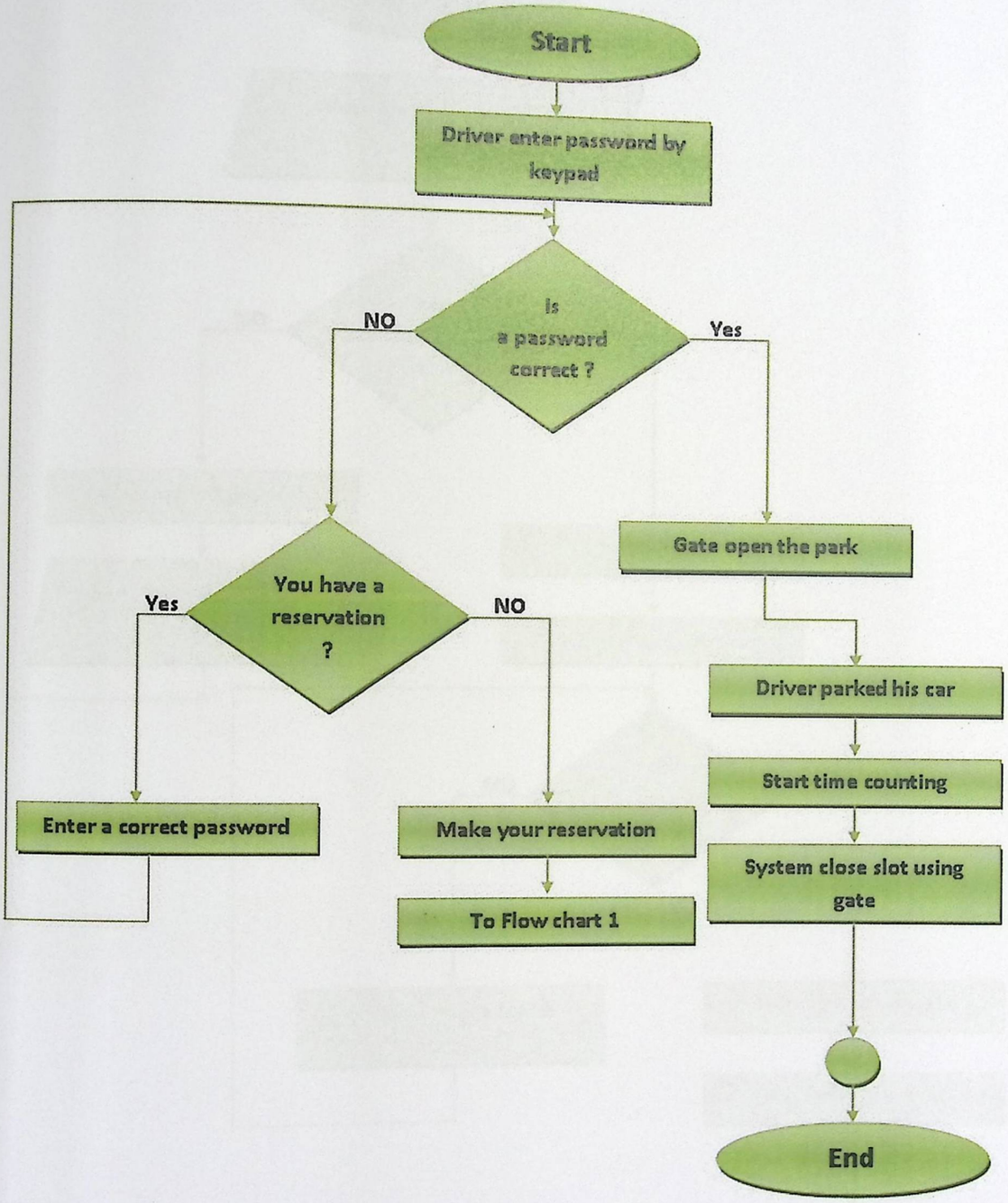
4.2.1Booking Flow Chart

This flow chart explain how the driver will booking the park? and what follow this booking?



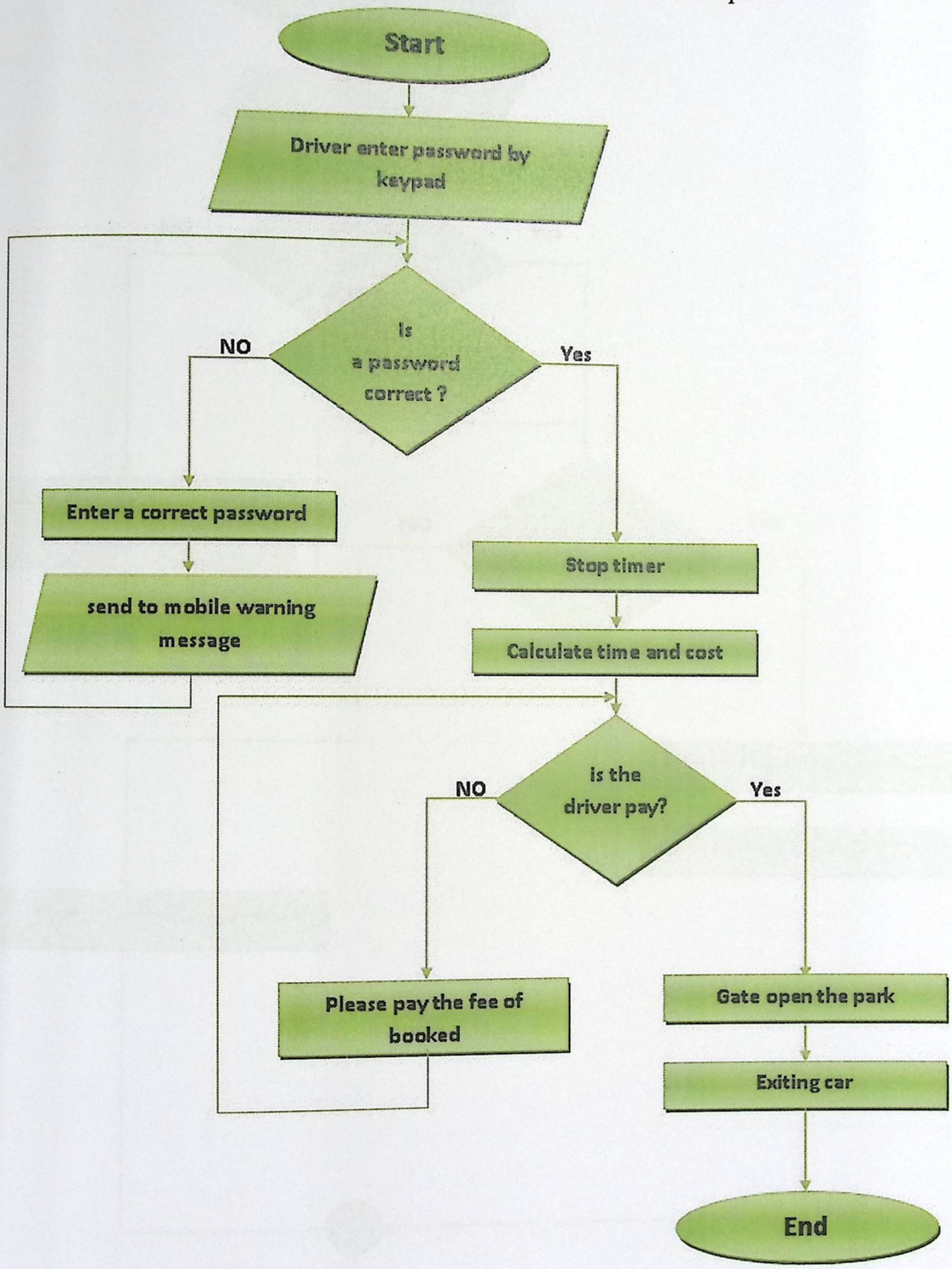
4.2.2 Entry Flow Chart

This flow chart explain how the driver will enter his car in the park? and what follow enter the park?



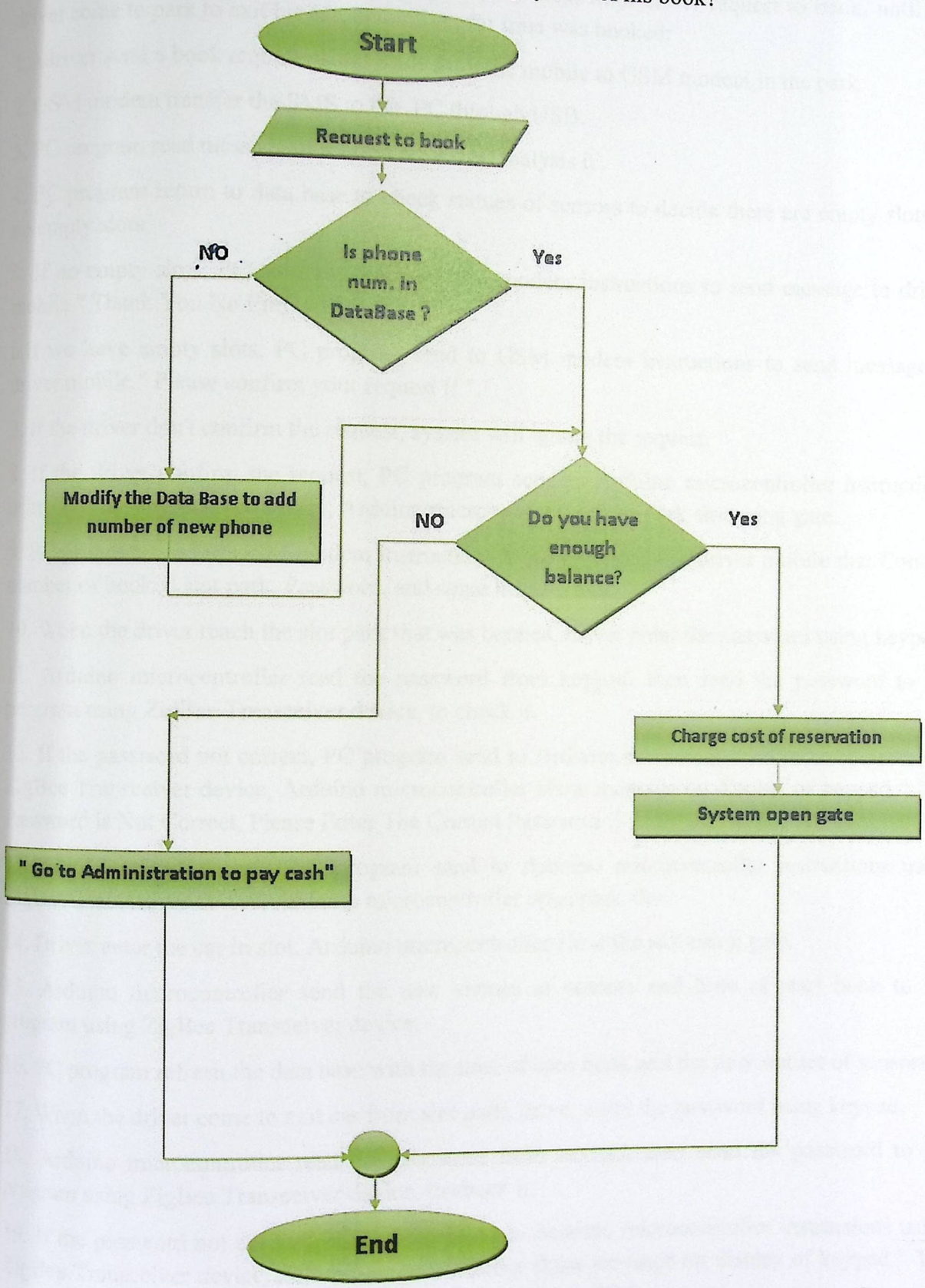
4.2.3 Exit Flow Chart

This flow chart explain how the driver will exit his car from the park?



4.2.4 Billing Flow Chart

This flow chart explain how the driver will pay cost of his book?



4.3 Step by Step Scenario

This scenario will show our project step by step, from driver request to book, until the driver come to park to exit his car, and billing for time was booked:

1.Adriver send a book request using SMS from the mobile to GSM modem in the park.

2. GSM modem transfer the SMS to the PC through USB.

3. PC program read message of request book and analysis it .

4. PC program return to data base to check statues of sensors to decide there are empty slots or no empty slots.

5. If no empty slots, PC program send to GSM modem instructions to send message to driver mobile " Thank You No Empty Park !! ".

6.If we have empty slots, PC program send to GSM modem instructions to send message to driver mobile " Please confirm your request !! ".

7. If the driver don't confirm the request, system will ignore the request.

8. If the driver confirm the request, PC program send to Arduino microcontroller instructions using ZigBee Transceiver device, Arduino microcontroller close park slot using gate.

9. PC program send to GSM modem instructions to send message to driver mobile that Contain number of booked slot park, Password, and some information.

10. When the driver reach the slot park that was booked, driver enter the password using keypad.

11. Arduino microcontroller read the password from keypad, then send the password to PC program using ZigBee Transceiver device, to check it.

12. If the password not correct, PC program send to Arduino microcontroller instructions using ZigBee Transceiver device, Arduino microcontroller show message on display of keypad " The Password is Not Correct, Please Enter The Correct Password !! ".

13.If the password correct, PC program send to Arduino microcontroller instructions using ZigBee Transceiver device, Arduino microcontroller open park slot.

14. Driver enter the car in slot, Arduino microcontroller close the slot using gate.

15. Arduino microcontroller send the new statues of sensors and time of start book to PC program using ZigBee Transceiver device.

16. PC program refresh the data base with the time of start book and the new statues of sensors.

17. When the driver come to exit car from slot park, driver enter the password using keypad.

18. Arduino microcontroller read the password from keypad, then send the password to PC program using ZigBee Transceiver device, to check it.

19. If the password not correct, PC program send to Arduino microcontroller instructions using ZigBee Transceiver device, Arduino microcontroller show message on display of keypad " The Password is Not Correct, Please Enter The Correct Password !! ".

20. If the password correct, PC program calculate time and cost.

21. PC program charge the cost by one of the method that will illustrate later.

22. PC program send to Arduino microcontroller instructions using ZigBee Transceiver device, Arduino microcontroller open park slot.

23. Driver exit the car from slot park.

24. Arduino microcontroller send the new statues of sensors to PC program using ZigBee Transceiver device.

25. PC program refresh the data base with the new statues of sensors.

Chapter Five

Chapter 5
System Implementation

In chapter 3, we discussed the conceptual design of the system in details including the idea, the main system parts, flow charts that describes their interaction with each other.

In this chapter we will dig more deeply in the implementation side of the project. Section 5.1 will focus on the electrical considerations in the design including the main circuit diagram, electrical calculations, and will discuss other possible variation of the design beside the advantages and disadvantages of each one. Section 5.2 will discuss the real methods, objects, and interfaces for software program.

5.1 Hardware system implementation

5.1.1 Detailed Circuit Design

The system consists of three types of nodes: mobile node, reference node and the coordinator node.

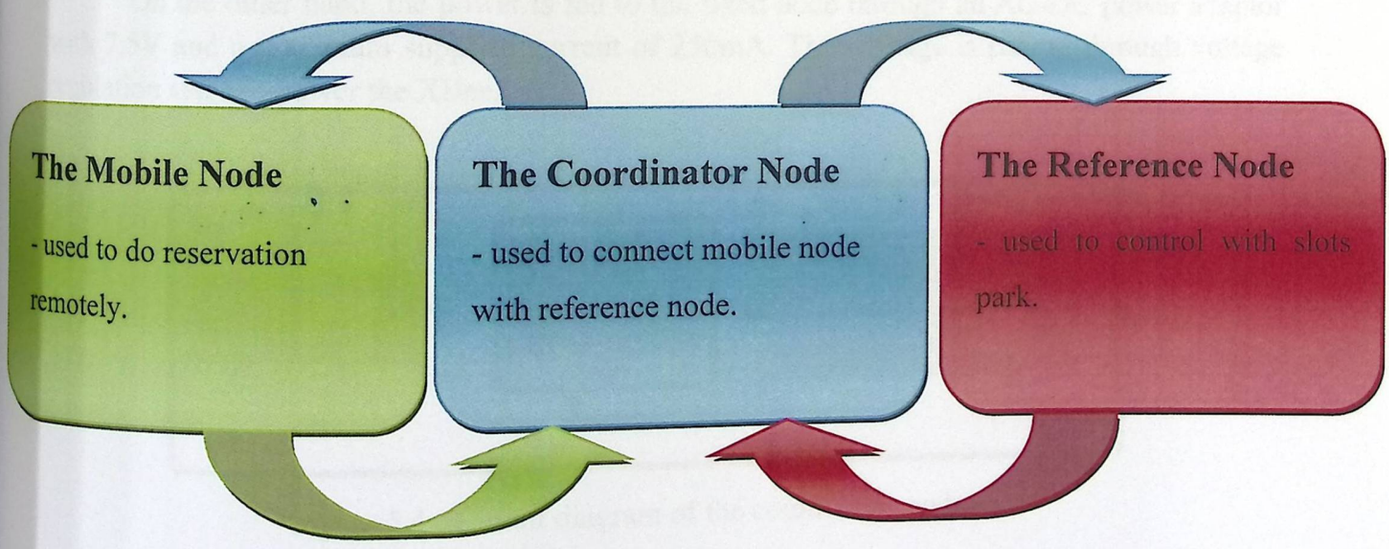


Figure 5.1 : General Block Diagram for Sub-Systems.

5.1.1.1 The Coordinator Node

The circuit of the coordinator is the simplest one, we used an XBee explorer dongle (shown in Figure 5.3) to connect the XBee to the administrator laptop. The dongle has two main functions: firstly it is used to interface the XBee to the laptop by converting serial data coming from XBee to the USB format (with proper voltage level translation) to be readable by the computer. Secondly, the dongle contains an inherited voltage regulator at 3.3V to power the XBee module. And we used an USB-GSM modem to connect the GSM modem (shown in Figure 5.2) to the administrator laptop.

Figure 5.2 : USB-GSM modem, PC together.

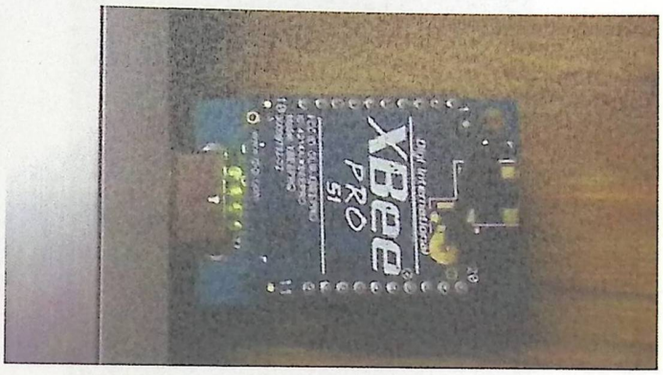


Figure 5.3 : XBee, PC and the dongle all together.

On the other hand, the power is fed to the fixed node through an AC-DC power adaptor with 7.5V and a maximum supplied current of 250mA. This voltage is passed through voltage regulation stage to power the XBee.

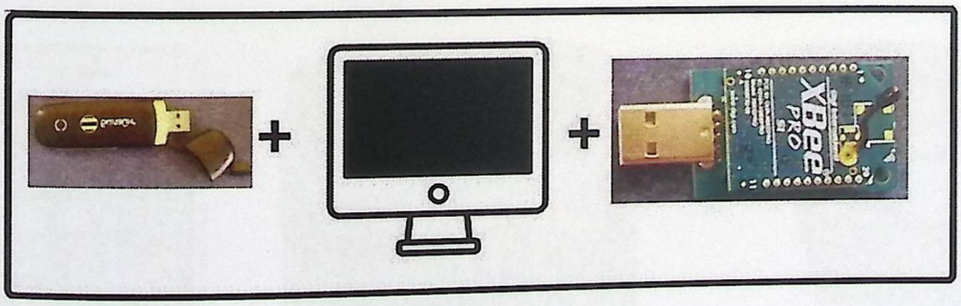


Figure 5.4 : Circuit diagram of the coordinator node.

The 1st component in the coordinator node is the XBee module, it is 20 pin transceiver with only four pins are of our interest which are Vcc, GND (power lines) and Dout, Din (serial lines).An additional optional pin will be used to indicate a packet reception on from the wireless link which is pin 6 (PWM0). In fact, through this pin XBee module generates a pulse width modulated (PWM) signal with the pulse frequency is varying according to the RSSI value of the received packet from the last hop.

جامعة بوليتكنك فلسطين
Palestine Polytechnic University
(PPU)
The Library المكتبة
Acc. 28848

5.1.1.1 The Coordinator Node

The circuit of the coordinator is the simplest one, we used an XBee explorer dongle (shown in Figure 5.3) to connect the XBee to the administrator laptop. The dongle has two main functions: firstly it is used to interface the XBee to the laptop by converting serial data coming from XBee to the USB format (with proper voltage level translation) to be readable by the computer. Secondly, the dongle contains an inherited voltage regulator at 3.3V to power the XBee module. And we used an USB-GSM modem to connect the GSM modem (shown in Figure 5.2) to the administrator laptop.

Figure 5.2 : USB-GSM modem, PC together.

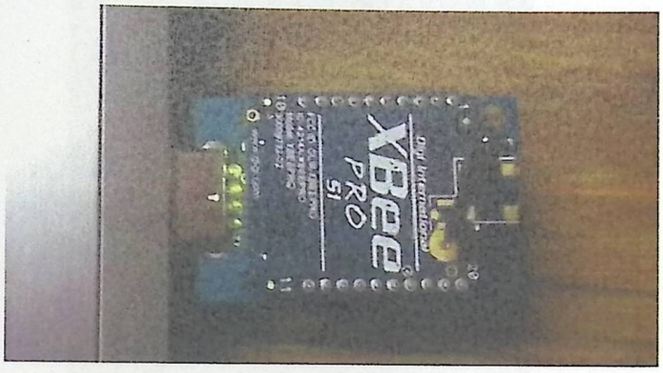


Figure 5.3 : XBee, PC and the dongle all together.

On the other hand, the power is fed to the fixed node through an AC-DC power adaptor with 7.5V and a maximum supplied current of 250mA. This voltage is passed through voltage regulation stage to power the XBee.

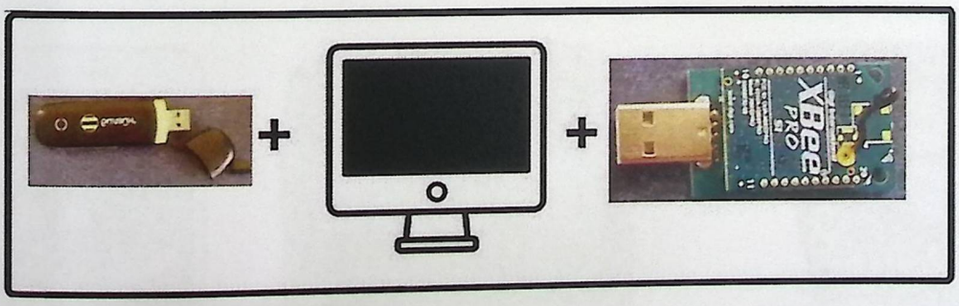


Figure 5 cuit diagram of the coordinator node.

Th or node is the XBee module, it is 20 pin transceiver with hich are Vcc, GND (power lines) and Dout, Din (serial indicate a packet reception on from the wireless this pin XBee module generates a pulse width ency is varying according to the RSSI value of the

جامعة بوليتكنك فلسطين
Palestine Polytechnic University
(PPU)
The Library المكتبة
Acc. 28848 الرقم

The 2nd component in the coordinator node is USB-GSM modem, it's transceiver SMS between the administrators laptop and the drivers mobiles, we have four types of messages: firstly, **Request Reservation Message** the direction of this message from mobile to laptop, the driver send the character " R " to reserve slot. Secondly, **Request to confirm** the direction of this message from laptop to mobile, laptop ask the driver to confirm request by send code as "4423". Thirdly, **Confirmation Message** the direction of this message from mobile to laptop, the driver send the code to confirm reservation. Finally, **Reservation Message** the direction of this message from laptop to mobile, laptop send number of reserved slot, password, and some information to driver.

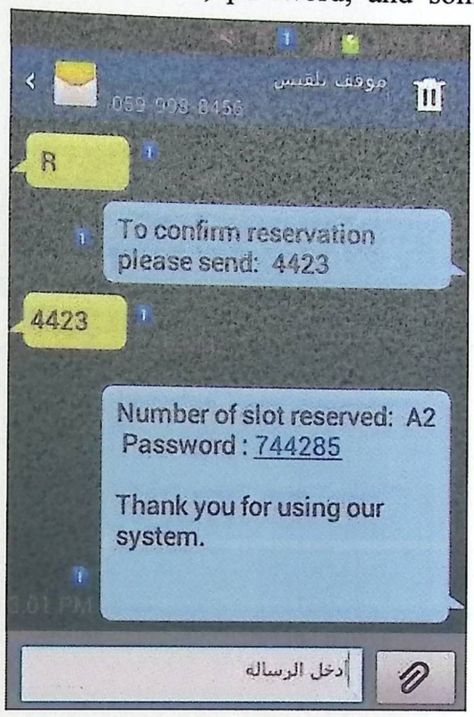


Figure 5.5 : Type of Messages.

5.1.1.2 The Mobile Node

In our system, we used the short message serves (SMS), that mean all type of phones can connect easily with our system.

Figure 5.6 : Type of Phones.

The 2nd component in the coordinator node is USB-GSM modem, it's role is to connect between the administrators laptop and the drivers mobiles, we have four types of messages firstly, **Request Reservation Message** the direction of this message from laptop to mobile, driver send the character " R " to reserve slot. Secondly, **Request Password Message** the direction of this message from laptop to mobile, laptop ask the driver to enter the password. Thirdly, **Confirmation Message** the direction of this message to connect between the coordinator send the code to confirm reservation. Finally, ID 2500 is microcontroller to control with the message from laptop to mobile, laptop send message to enable to driver to enter the password information to driver.

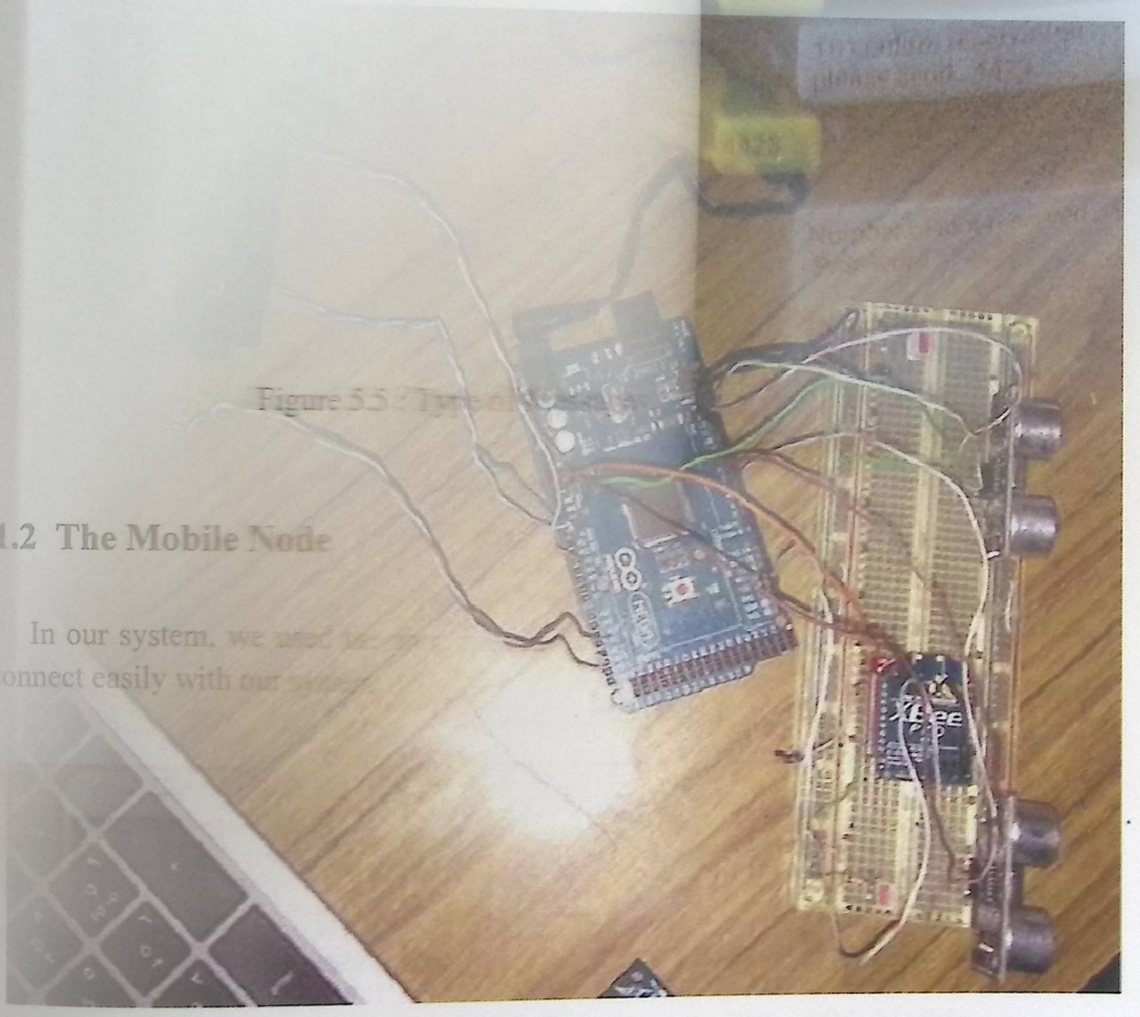


Figure 5.7 : The Reference Node.

5.1.1.2 The Reference Node

XBee Pro Module, Arduino Mega 2560, Ultrasonic Sensor, LCD (16 X 2), and Keypad all there component together are Reference Node shown in Figure 5.7.

We used XBee Pro Module as transceiver device to connect between the coordinator node and the reference node. We used Arduino Mega 2560 as microcontroller to control with the ultrasonic sensor, LCD, and Keypad. We used Keypad to enable to driver to enter the password to open the reserved slot.

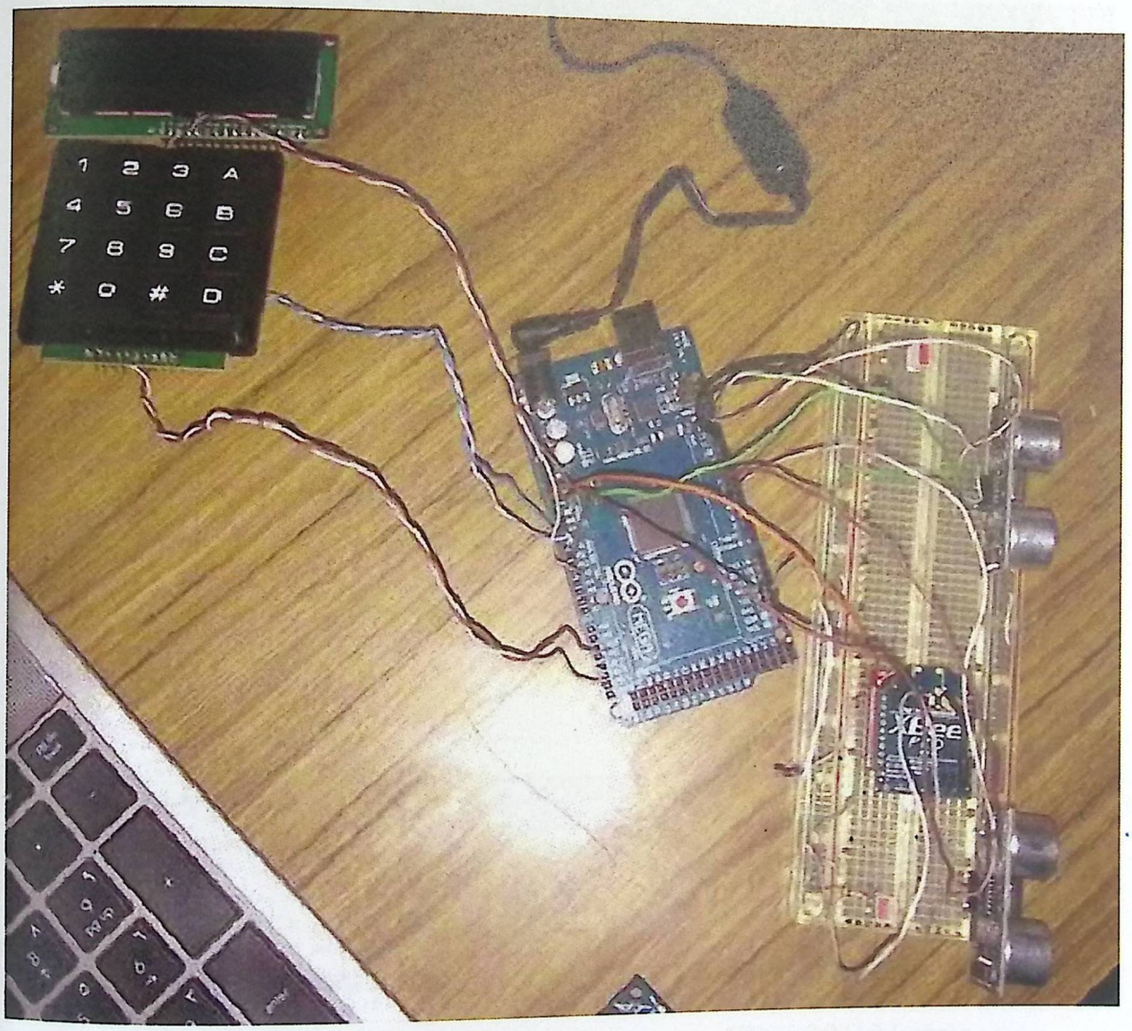


Figure 5.7 : The Reference Node.

5.2Software system implementation

We will discuss the real methods, objects, and interfaces for software program.

5.2.1 Main Interface

This is the main interface for our program, the program doesn't operate before connect XBee PRO Module and USB-GSM Modem as shown in Figure 5.8, the module and modem are not connect because that the tabs of program doesn't appear to use.

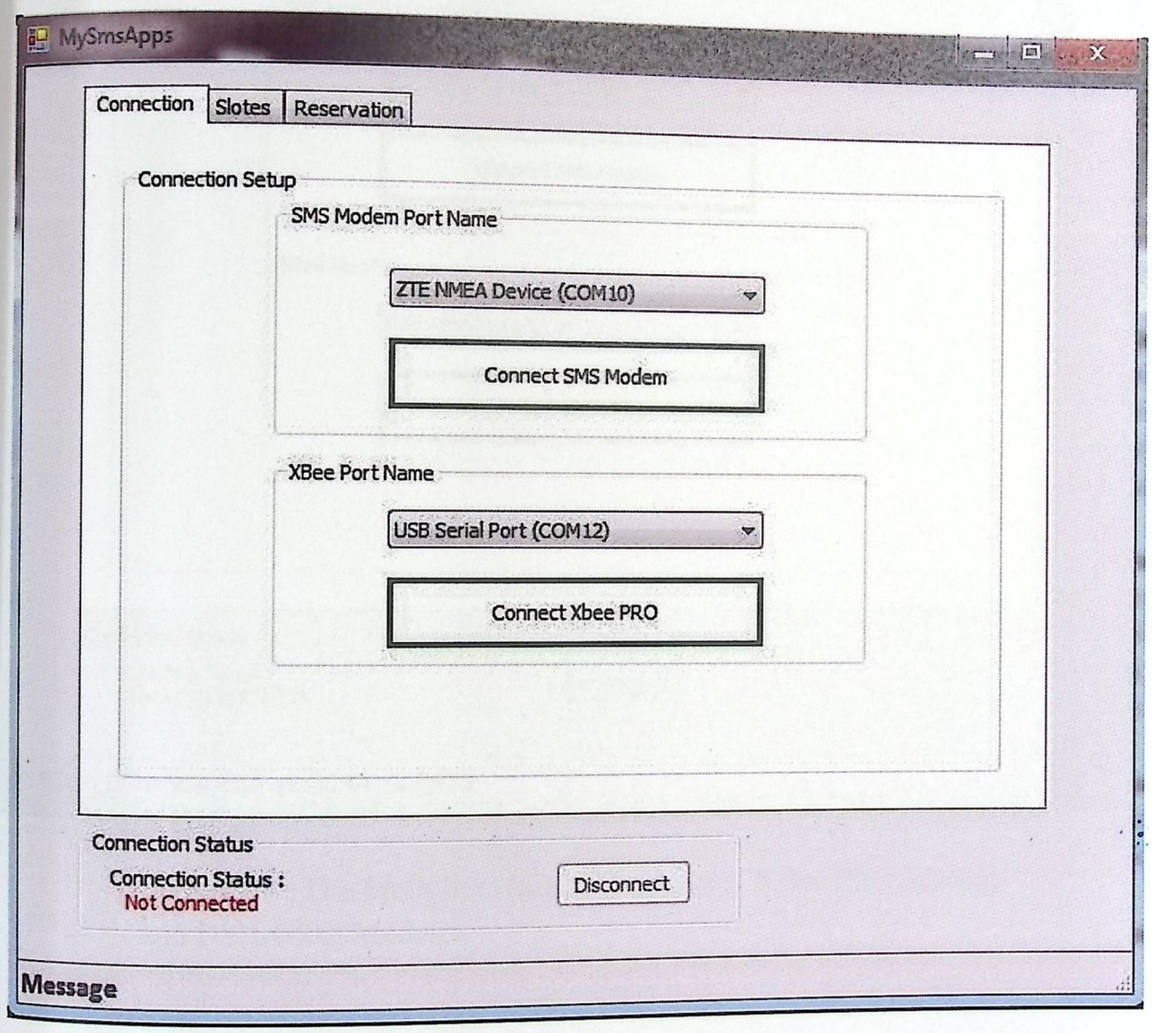


Figure 5.8 : The Main Interface before connect XBee PRO Module and USB-GSM Modem.

The program does operate after connect XBee PRO Module and USB-GSM Modem as shown in Figure 5.9 the module and modem are connect because that the tabs of program does appear to use.

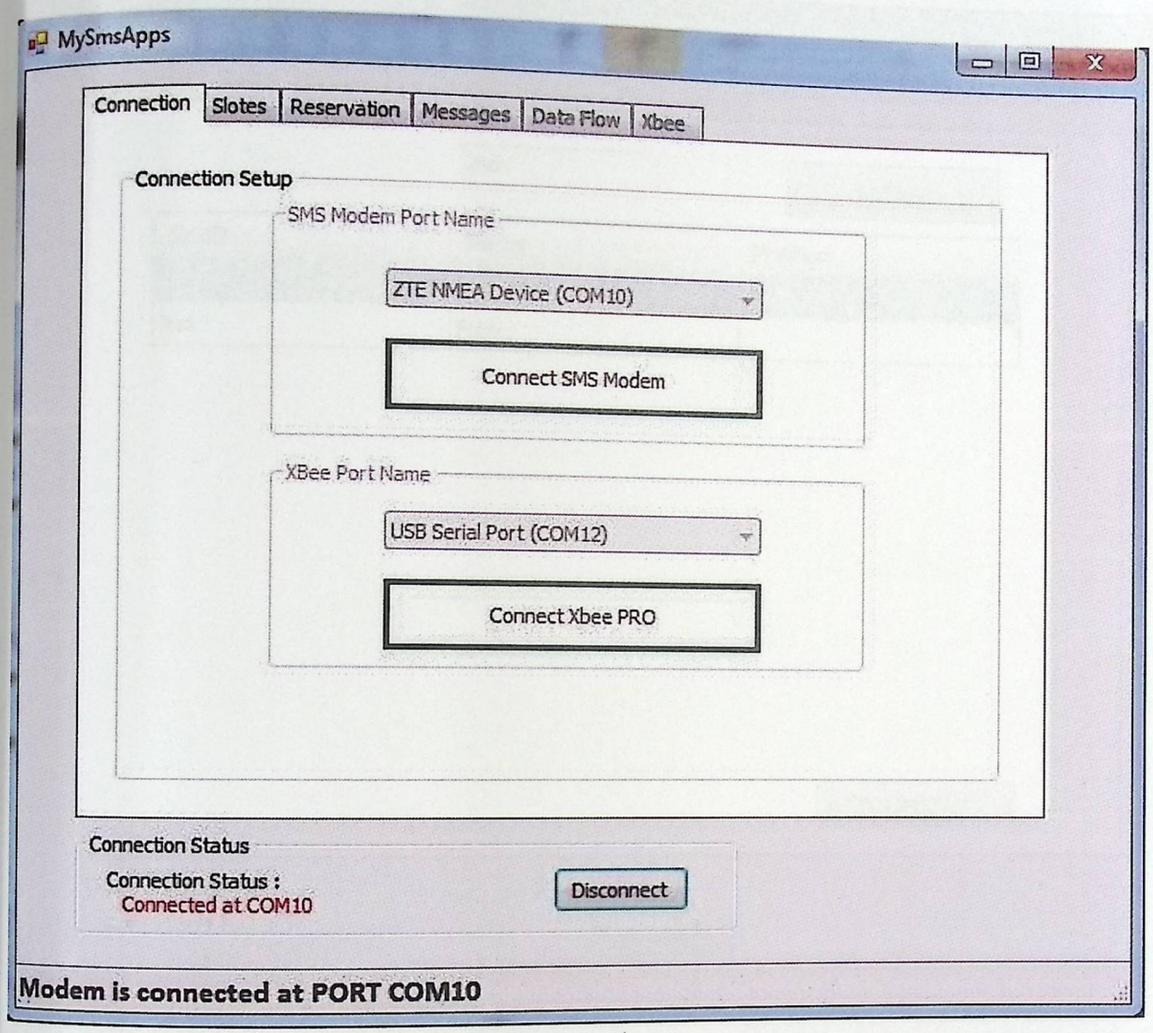


Figure 5.9 : The Main Interface after connect XBee PRO Module and USB-GSM Modem.

5.2.2Slots Interface

In this interface the administrator can monitor the status of slots, **free** when the slot doesn't reserved, **reserved** when the slot does reserved, **full** when the driver enter the car in slot.

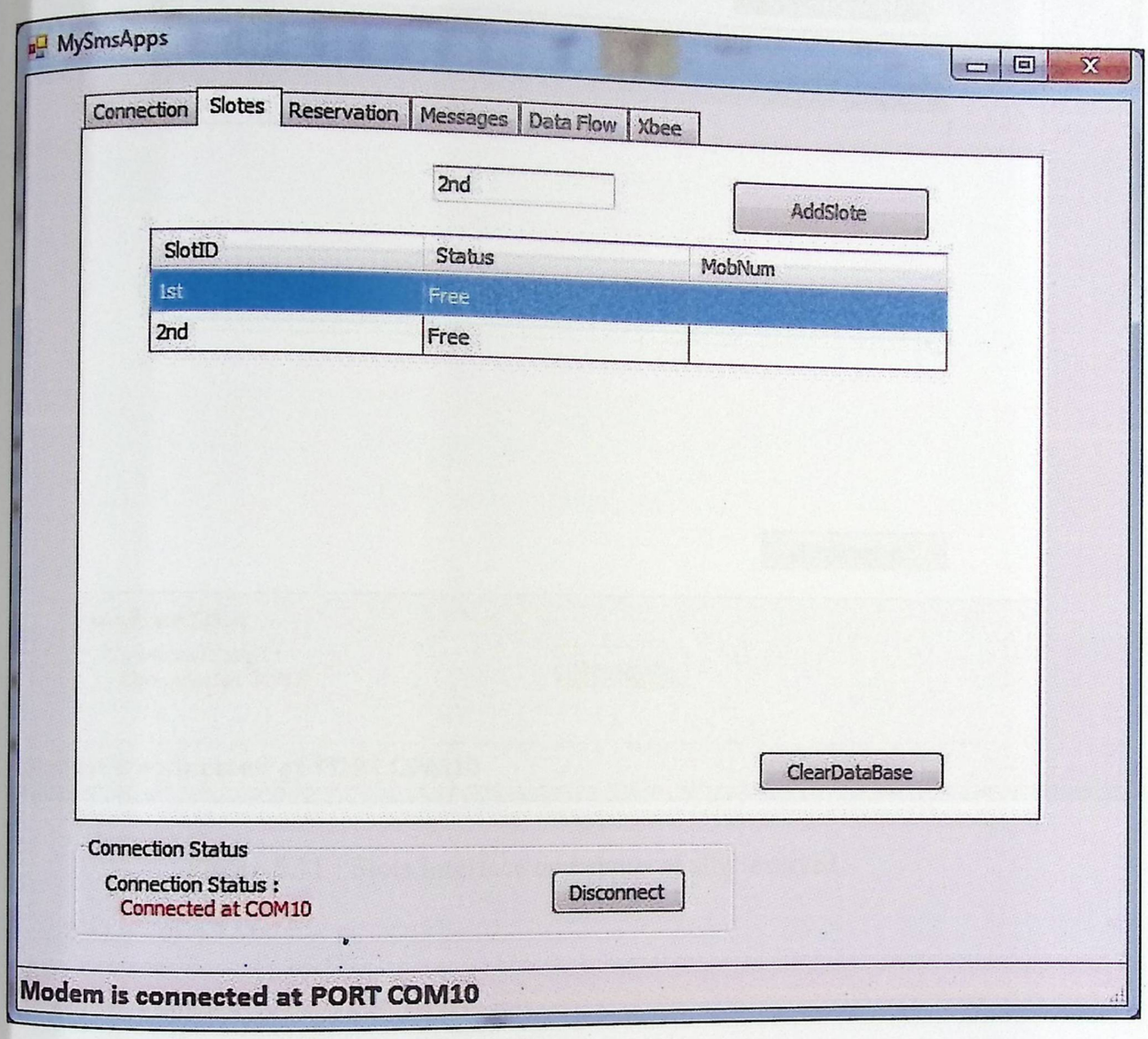


Figure 5.10 : Slots Interface and status of slots are free.

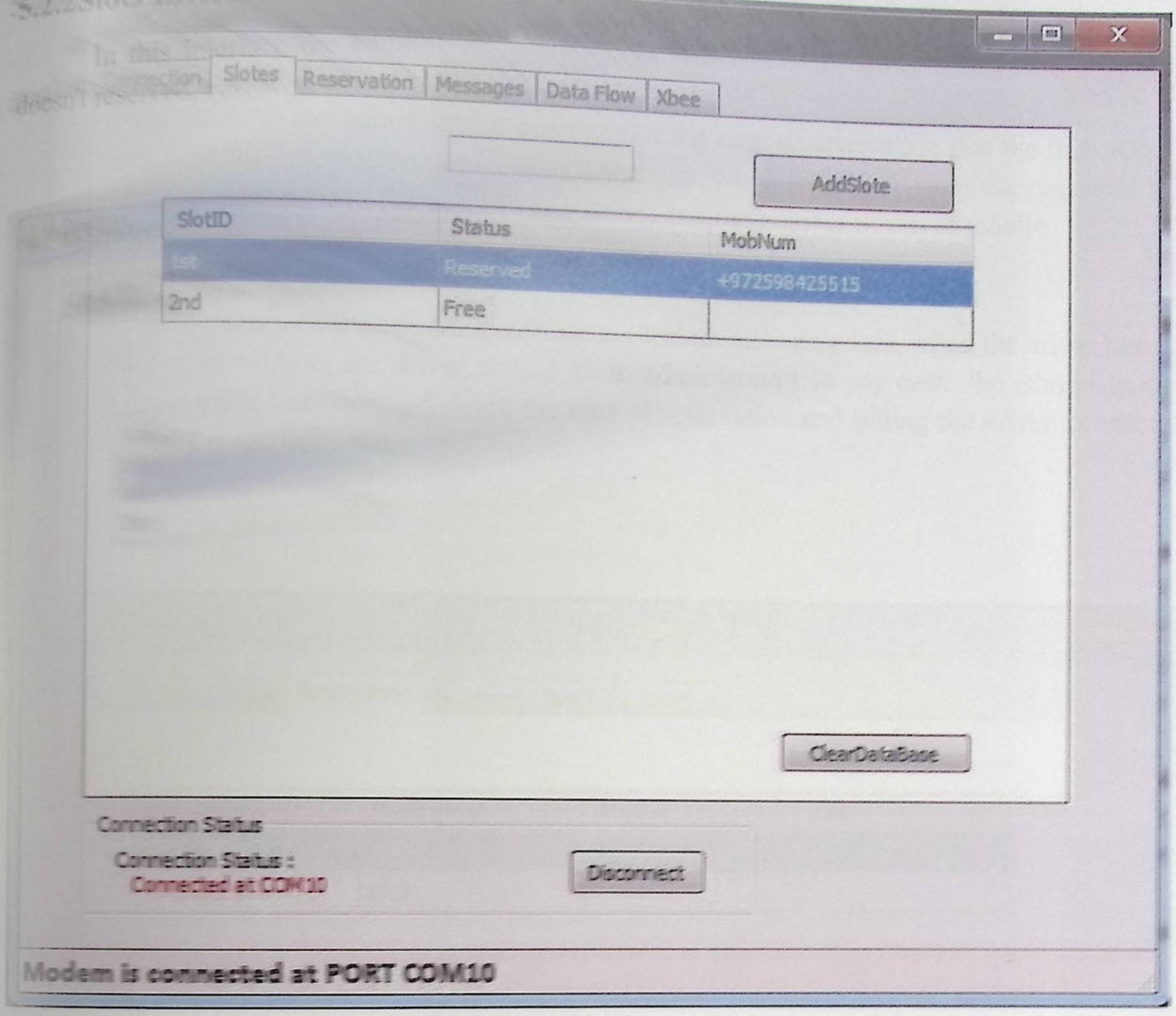


Figure 5.11 : Slots Interface and status of slot reserved.

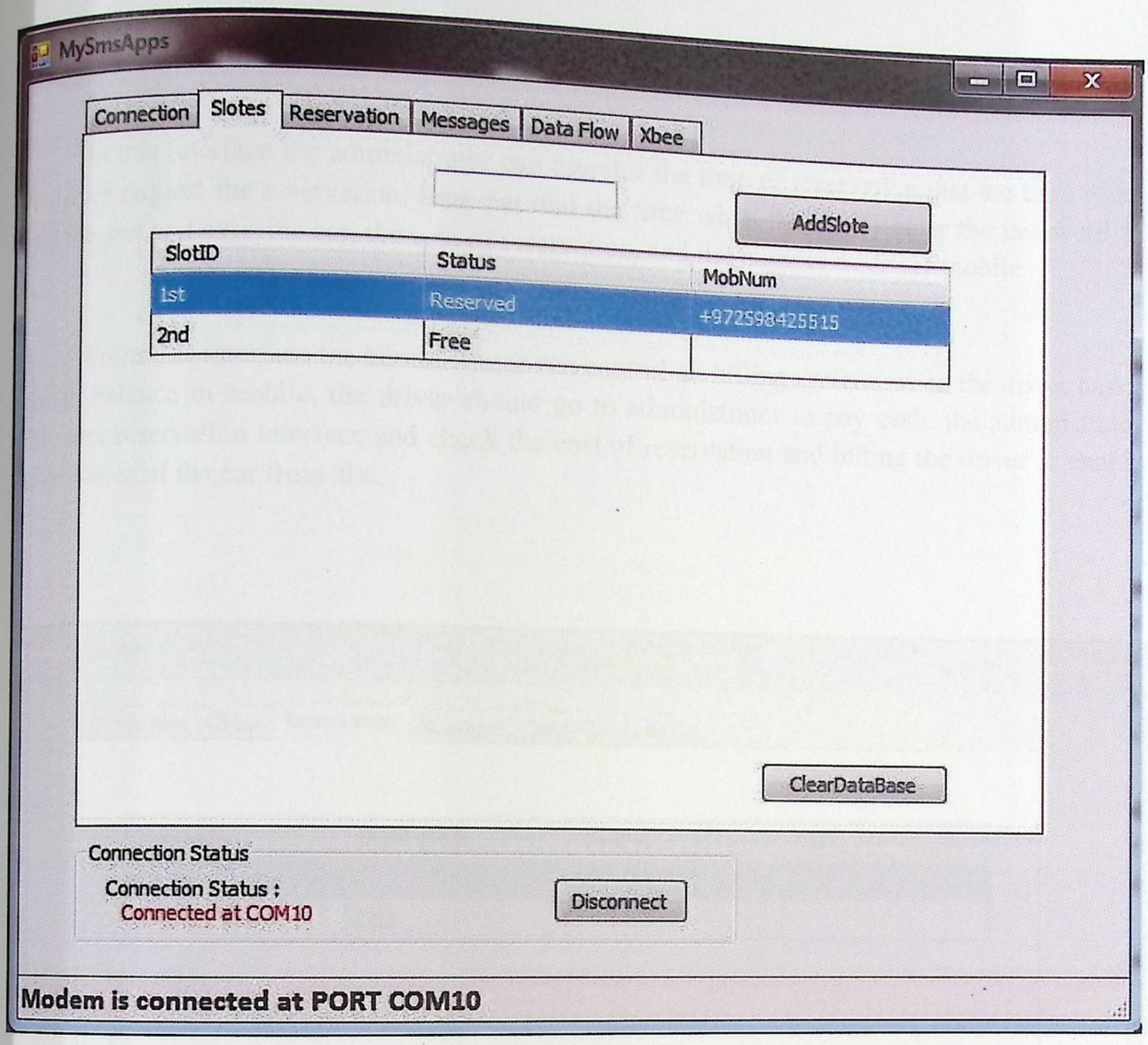


Figure 5.11 : Slots Interface and status of slot reserved.

5.2.3Reservation Interface

In this interface the administrator can monitor the time of reservation that the time when the driver request the reservation, time out that the time when the driver enter the password to open the slot and exist the car, the cost of reservation, and the number of driver mobile.

From this interface the administrator can control on billing system, when the driver hasn't enough balance in mobile, the driver should go to administrator to pay cash, the administrator will enter reservation interface and check the cost of reservation and billing the driver to enable the driver exist the car from slot.

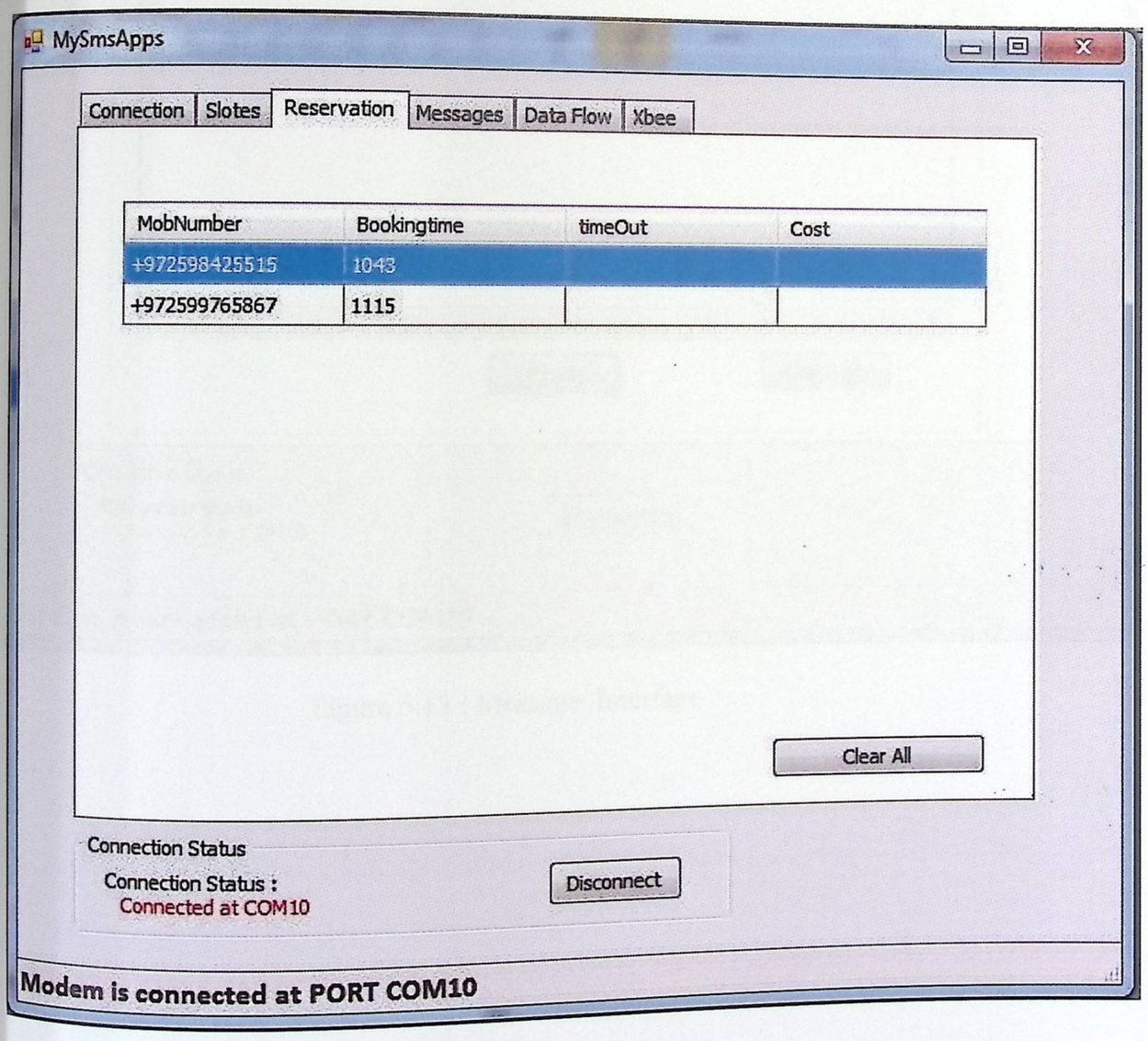


Figure 5.12 : Reservation Interface.

5.2.4Message Interface

In this interface the administrator can monitor the messages of reservation, messages of confirmation reservation, and the number of driver mobile.

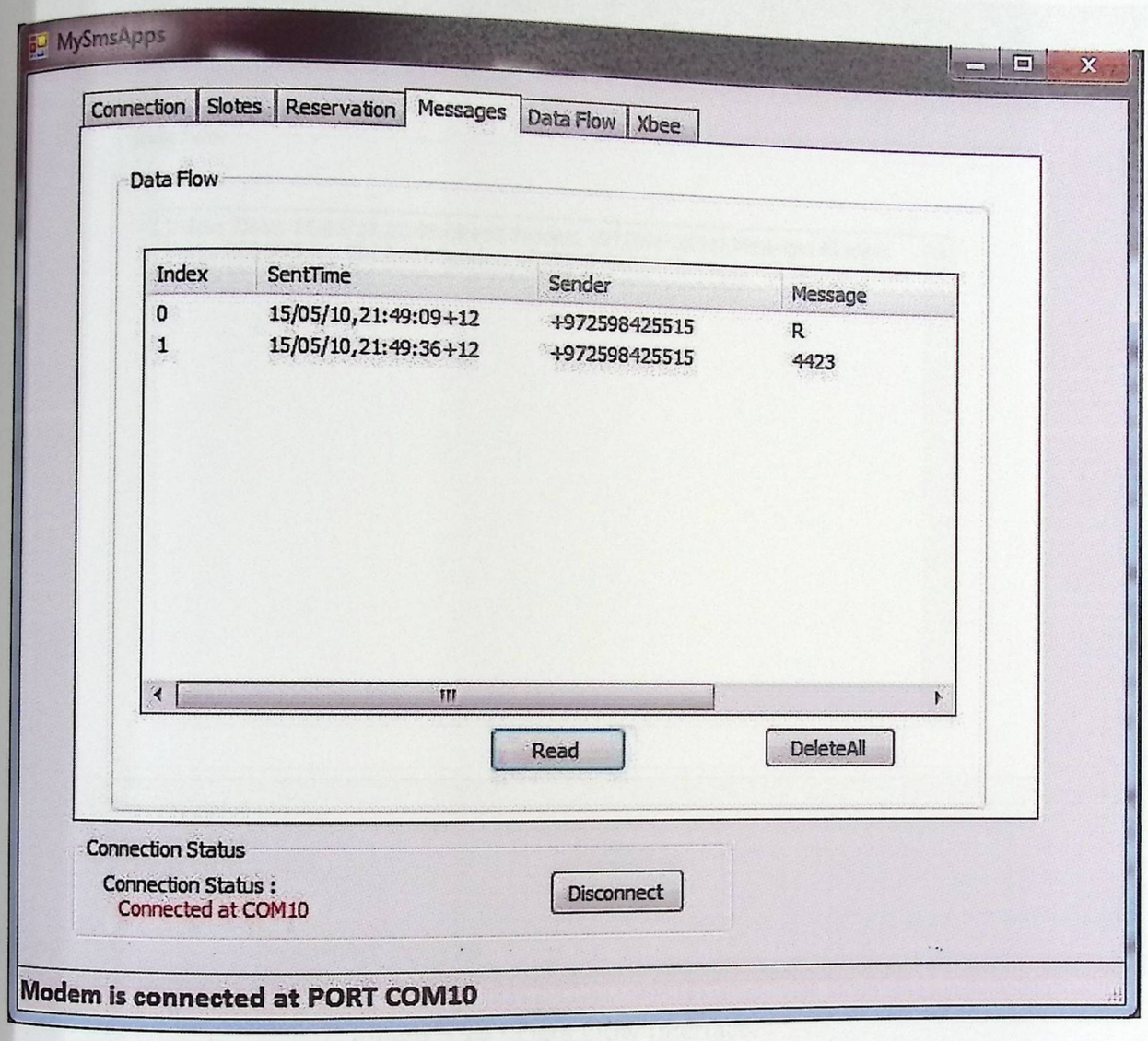


Figure 5.13 : Message Interface.

5.2.5Data Flow Interface

This interface display all message reach the USB-GSM Modem.

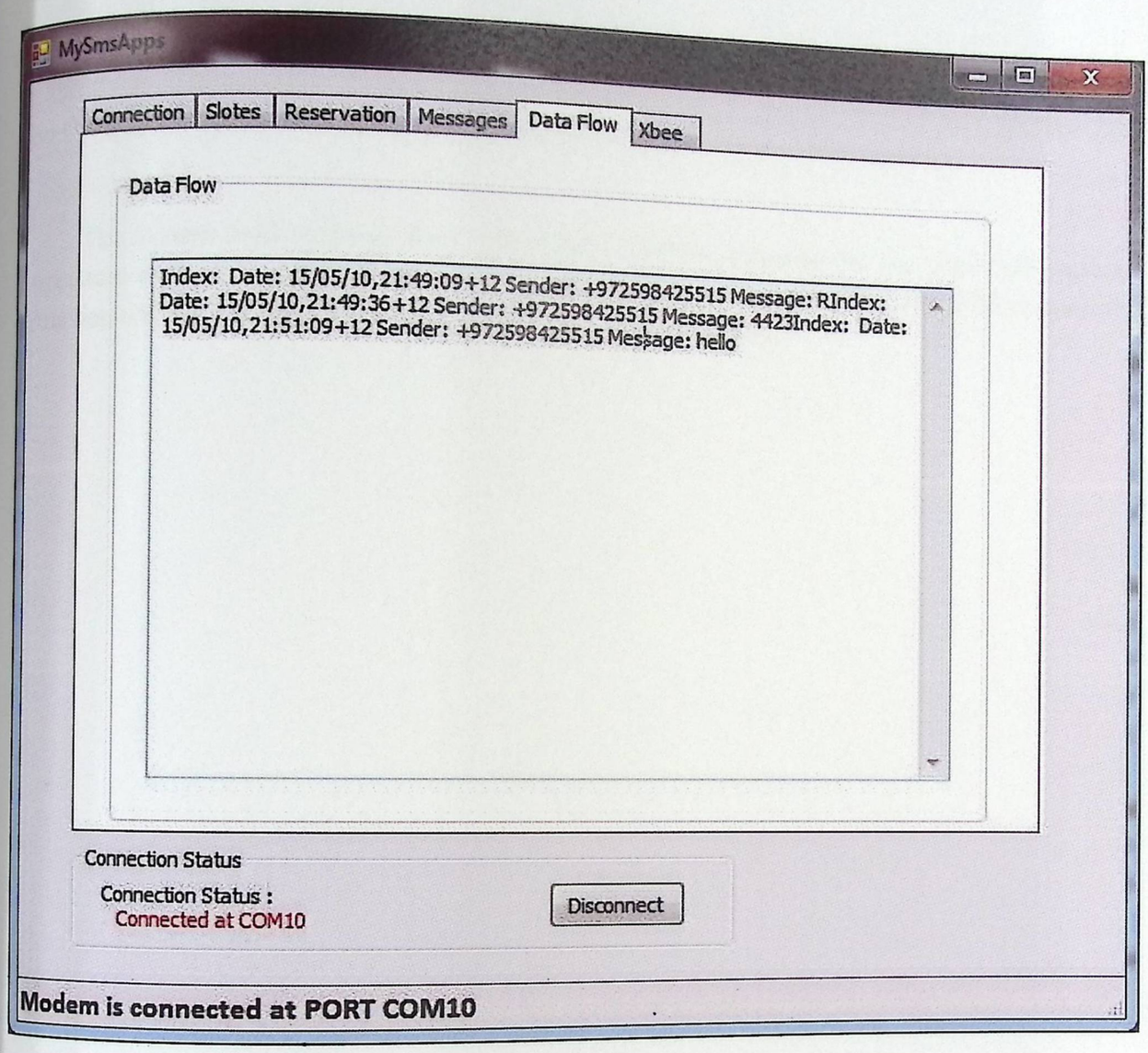


Figure 5.14 : Data Flow Interface.

5.3 Conclusion

The system designed to enable the driver reserve slot in park remotely using mobile, then go directly to reserved slot, all that decrease the traffic in the street and decrease the theft car. Also Our system was implemented to help the driver to save time, effort and cost.

The system will work in door better than outdoor, because the sun rays will increase temperature of Arduino that effect on speed of processing, that mean the speed of execute the instruction will be low, we need more time to do some instruction.

5.4 Recommendations

For the future works, we recommended the following:

1. Try to use extended range XBee modules to reduce the cost even more.
2. Make android application to become available.
3. Use GPS to determined the better park for the locution of driver.
4. Execute the system in multi floor and connect all systems.
5. Will discount the fee of reservation by android application.

Appendix A

XBee®/XBee-PRO® ZB RF Modules

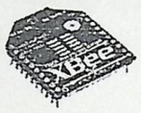
ZigBee RF Modules by Digi International

Models: XBEE2, XBEEPRO2, PRO S2B

Hardware: S2 and S2B

Firmware Versions:

- 20xx - Coordinator - AT/Transparent Operation
- 21xx - Coordinator - API Operation
- 22xx - Router - AT/Transparent Operation
- 23xx - Router - API Operation
- 28xx - End Device - AT/Transparent Operation
- 29xx - End Device - API Operation



Digi International Inc.
11001 Bren Road East
Minnetonka, MN 55343
877 912-3444 or 952 912-3444
http://www.digi.com

90000976_K
March 2012

© 2012 Digi International, Inc. All rights reserved

No part of the contents of this manual may be transmitted or reproduced in any form or by any means without the written permission of Digi International, Inc.

ZigBee® is a registered trademark of the ZigBee Alliance.

XBee® and XBee-PRO® are registered trademarks of Digi International, Inc.

Technical Support: Phone: (866) 765-9885 toll-free U.S.A. & Canada
(801) 765-9885 Worldwide
8:00 am - 5:00 pm [U.S. Mountain Time]

Online Support: http://www.digi.com/support/eservice/login.jsp

Email: rf-experts@digi.com

4. Transmission, Addressing, and Routing

Addressing

All ZigBee devices have two different addresses, a 64-bit and a 16-bit address. The characteristics of each are described below.

64-bit Device Addresses

The 64-bit address is a unique device address assigned during manufacturing. This address is unique to each physical device. The 64-bit address includes a 3-byte Organizationally Unique Identifier (OUI) assigned by the IEEE. The 64-bit address is also called the extended address.

16-bit Device Addresses

A device receives a 16-bit address when it joins a ZigBee network. For this reason, the 16-bit address is also called the "network address". The 16-bit address of 0x0000 is reserved for the coordinator. All other devices receive a randomly generated address from the router or coordinator device that allows the join. The 16-bit address can change under certain conditions:

- An address conflict is detected where two devices are found to have the same 16-bit address
- A device leaves the network and later joins (it can receive a different address)

All ZigBee transmissions are sent using the source and destination 16-bit addresses. The routing tables on ZigBee devices also use 16-bit addresses to determine how to route data packets through the network. However, since the 16-bit address is not static, it is not a reliable way to identify a device.

To solve this problem, the 64-bit destination address is often included in data transmissions to guarantee data is delivered to the correct destination. The ZigBee stack can discover the 16-bit address, if unknown, before transmitting data to a remote.

Application Layer Addressing

ZigBee devices can support multiple application profiles, cluster IDs, and endpoints. (See "ZigBee Application Layers - In Depth" in chapter 3.) Application layer addressing allows data transmissions to be addressed to specific profile IDs, cluster IDs, and endpoints. Application layer addressing is useful if an application must

- Interoperate with other ZigBee devices outside of the Digi application profile
- Utilize service and network management capabilities of the ZDO
- Operate on a public application profile such as Home Controls or Smart Energy.

The API firmware provides a simple yet powerful interface that can easily send data to any profile ID, endpoint, and cluster ID combination on any device in a ZigBee network.

Data Transmission

ZigBee data packets can be sent as either unicast or broadcast transmissions. Unicast transmissions route data from one source device to one destination device, whereas broadcast transmissions are sent to many or all devices in the network.

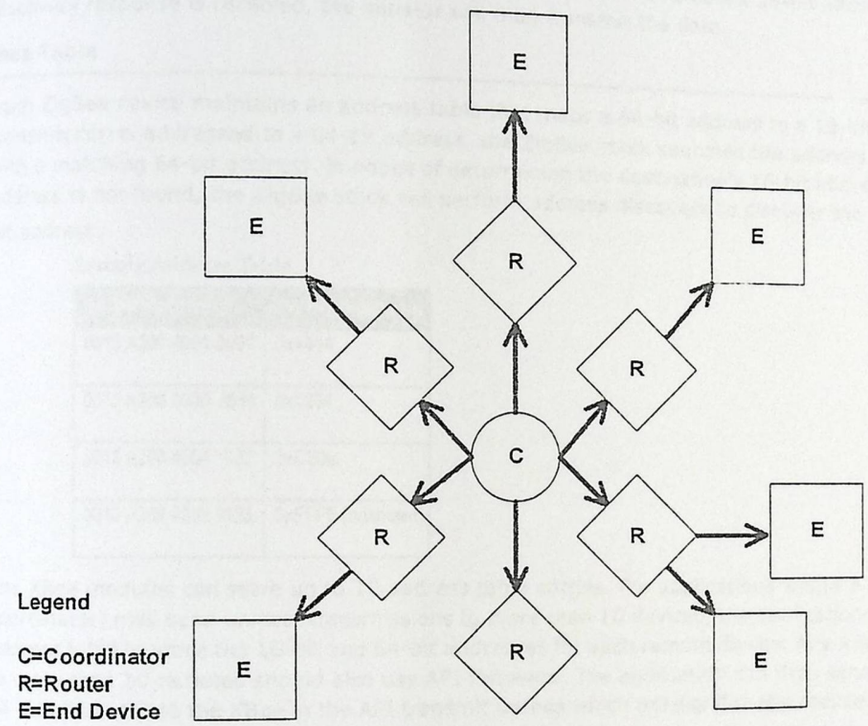
© 2012 Digi International, Inc.

Broadcast Transmissions

Broadcast transmissions within the ZigBee protocol are intended to be propagated throughout the entire network such that all nodes receive the transmission. To accomplish this, the coordinator and all routers that receive a broadcast transmission will retransmit the packet three times.

Note: when a router or coordinator delivers a broadcast transmission to an end device child, the transmission is only sent once (immediately after the end device wakes and polls the parent for any new data). See Parent Operation section in chapter 6 for details.

Broadcast Data Transmission



Each node that transmits the broadcast will also create an entry in a local broadcast transmission table. This entry is used to keep track of each received broadcast packet to ensure the packets are not endlessly transmitted. Each entry persists for 8 seconds. The broadcast transmission table holds 8 entries.

For each broadcast transmission, the ZigBee stack must reserve buffer space for a copy of the data packet. This copy is used to retransmit the packet as needed. Large broadcast packets will require more buffer space. This information on buffer space is provided for general knowledge; the user does not and cannot change any buffer spacing. Buffer spacing is handled automatically by the XBee module.

Since broadcast transmissions are retransmitted by each device in the network, broadcast messages should be used sparingly.

Unicast Transmissions

Unicast transmissions are sent from one source device to another destination device. The destination device could be an immediate neighbor of the source, or it could be several hops away. Unicast transmissions that are sent along a multiple hop path require some means of establishing a route to the destination device. See the "RF Packet Routing" section in chapter 4 for details.

Address Resolution

As mentioned previously, each device in a ZigBee network has both a 16-bit (network) address and a 64-bit (extended) address. The 64-bit address is unique and assigned to the device during manufacturing, and the

© 2013 Digi International, Inc.

16-bit address is obtained after joining a network. The 16-bit address can also change under certain conditions.

When sending a unicast transmission, the ZigBee network layer uses the 16-bit address of the destination and each hop to route the data packet. If the 16-bit address of the destination is not known, the ZigBee stack includes a discovery provision to automatically discover the destination device's 16-bit address before routing the data.

To discover a 16-bit address of a remote, the device initiating the discovery sends a broadcast address discovery transmission. The address discovery broadcast includes the 64-bit address of the remote device whose 16-bit address is being requested. All nodes that receive this transmission check the 64-bit address in the payload and compare it to their own 64-bit address. If the addresses match, the device sends a response packet back to the initiator. This response includes the remote's 16-bit address. When the discovery response is received, the initiator will then transmit the data.

Address Table

Each ZigBee device maintains an address table that maps a 64-bit address to a 16-bit address. When a transmission is addressed to a 64-bit address, the ZigBee stack searches the address table for an entry with a matching 64-bit address, in hopes of determining the destination's 16-bit address. If a known 16-bit address is not found, the ZigBee stack will perform address discovery to discover the device's current 16-bit address.

Sample Address Table

64-bit Address	16-bit Address
0013 A200 4000 0001	0x4414
0013 A200 400A 3568	0x1234
0013 A200 4004 1122	0xC200
0013 A200 4002 1123	0xFFFE (unknown)

The XBee modules can store up to 10 address table entries. For applications where a single device (e.g. coordinator) may send unicast transmissions to more than 10 devices, the application should implement an address table to store the 16-bit and 64-bit addresses for each remote device. Any XBee that will send data to more than 10 remotes should also use API firmware. The application can then send both the 16-bit and 64-bit addresses to the XBee in the API transmit frames which will significantly reduce the number of 16-bit address discoveries and greatly improve data throughput.

If an application will support an address table, the size should ideally be larger than the maximum number of destination addresses the device will communicate with. Each entry in the address table should contain a 64-bit destination address and its last known 16-bit address.

When sending a transmission to a destination 64-bit address, the application should search the address table for a matching 64-bit address. If a match is found, the 16-bit address should be populated into the 16-bit address field of the API frame. If a match is not found, the 16-bit address should be set to 0xFFFE (unknown) in the API transmit frame.

The API provides indication of a remote device's 16-bit address in the following frames:

- All receive data frames

  Rx Data (0x90)

  Rx Explicit Data (0x91)

  IO Sample Data (0x92)

  Node Identification Indicator (0x95)

  Route Record Indicator (0xA1)

  etc.

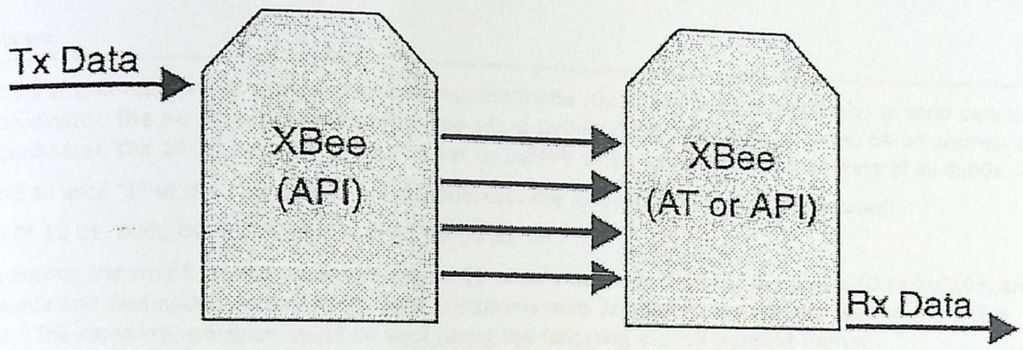
- Transmit status frame (0x8B)

©2012 Digi International, Inc.

The application should always update the 16-bit address in the address table when one of these frames is received to ensure the table has the most recently known 16-bit address. If a transmission failure occurs, the application should set the 16-bit address in the table to 0xFFFE (unknown).

Fragmentation

Each unicast transmission may support up to 84 bytes of RF payload. (Enabling security or using source routing can reduce this number. See the NP command for details.) However, the XBee ZB firmware supports a ZigBee feature called fragmentation that allows a single large data packet to be broken up into multiple RF transmissions and reassembled by the receiver before sending data out its UART. This is shown in the image below.



The API transmit frame can include up to 255 bytes of data, which will be broken up into multiple transmissions and reassembled on the receiving side. If one or more of the fragmented messages are not received by the receiving device, the receiver will drop the entire message, and the sender will indicate a transmission failure in the Tx Status API frame. ZB firmware can only support one fragmented packet at a time at the receive node, due to memory constraints.

Applications that do not wish to use fragmentation should avoid sending more than the maximum number of bytes in a single RF transmission. See the "Maximum RF Payload Size" section for details.

Devices will not receive or reassemble fragmented RF packets if RTS flow control is enabled (D6 command).

Data Transmission Examples

AT Firmware

To send a data packet in AT firmware, the DH and DL commands must be set to match the 64-bit address of the destination device. DH must match the upper 4-bytes, and DL must match the lower 4 bytes. Since the coordinator always receives a 16-bit address of 0x0000, a 64-bit address of 0x0000000000000000 is defined as the coordinator's address (in ZB firmware). The default values of DH and DL are 0x00, which sends data to the coordinator.

Example 1: Send a transmission to the coordinator.

(In this example, a '\r' refers to a carriage return character.)

A router or end device can send data in two ways. First, set the destination address (DH and DL commands) to 0x00.

1. Enter command mode ('+++')
2. After receiving an OK\r, issue the following commands:
   a. ATDH0\r
   b. ATDL0\r
   c. ATCN\r
3. Verify that each of the 3 commands returned an OK\r response.
4. After setting these command values, all serial characters will be sent as a unicast transmission to the coordinator.

© 2012 Digi International, Inc.

Alternatively, if the coordinator's 64-bit address is known, DH and DL can be set to the coordinator's 64-bit address. Suppose the coordinator's address is 0x0013A200404A2244.

1. Enter command mode ('+++')
2. After receiving an OK\r, issue the following commands:
   a. ATDH13A200\r
   b. ATDL404A2244\
   c. ATCN\r
3. Verify that each of the 3 commands returned an OK\r response.
4. After setting these command values, all serial characters will be sent as a unicast transmission to the coordinator.

API Firmware

Use the transmit request, or explicit transmit request frame (0x10 and 0x11 respectively) to send data to the coordinator. The 64-bit address can either be set to 0x0000000000000000, or to the 64-bit address of the coordinator. The 16-bit address should be set to 0xFFFE when using the 64-bit address of all 0x00s.

To send an ascii "1" to the coordinator's 0x00 address, the following API frame can be used:

7E 00 0F 10 01 0000 0000 0000 0000 FFFE 00 00 31 C0

If the explicit transmit frame is used, the cluster ID should be set to 0x0011, the profile ID to 0xC105, and the source and destination endpoints to 0xE8 (recommended defaults for data transmissions in the Digi profile.) The same transmission could be sent using the following explicit transmit frame:

7E 00 15 11 01 0000 0000 0000 0000 FFFE E8 E8 0011 C105 00 00 31 18

Notice the 16-bit address is set to 0xFFFE. This is required when sending to a 64-bit address of 0x00s.

Now suppose the coordinator's 64-bit address is 0x0013A200404A2244. The following transmit request API frame (0x10) will send an ASCII "1" to the coordinator:

7E 00 0F 10 01 0013 A200 404A 2244 0000 0000 31 18

Example 2: Send a broadcast transmission.

(In this example, a '\r' refers to a carriage return character.)

Perform the following steps to configure a broadcast transmission:

1. Enter command mode ('+++')
2. After receiving an OK\r, issue the following commands:
   a. ATDH0\r
   b. ATDLffff\r
   c. ATCN\r
3. Verify that each of the 3 commands returned an OK\r response
4. After setting these command values, all serial characters will be sent as a broadcast transmission.

API Firmware

This example will use the transmit request API frame (0x10) to send an ASCII "1" in a broadcast transmission.

To send an ascii "1" as a broadcast transmission, the following API frame can be used:

7E 00 0F 10 01 0000 0000 0000 FFFF FFFE 00 00 31 C2

Notice the destination 16-bit address is set to 0xFFFE for broadcast transmissions.

RF Packet Routing

Unicast transmissions may require some type of routing. ZigBee includes several different ways to route data, each with its own advantages and disadvantages. These are summarized in the table below.

© 2012 Digi International, Inc.

Routing Approach	Description	When to Use
Ad hoc On-demand Distance Vector (AODV) Mesh Routing	Routing paths are created between source and destination, possibly traversing multiple nodes ("hops"). Each device knows who to send data to next to eventually reach the destination	Use in networks that will not scale beyond about 40 destination devices.
Many-to-One Routing	A single broadcast transmission configures reverse routes on all devices into the device that sends the broadcast	Useful when many remote devices send data to a single gateway or collector device.
Source Routing	Data packets include the entire route the packet should traverse to get from source to destination	Improves routing efficiency in large networks (over 40 remote devices)

Note - End devices do not make use of these routing protocols. Rather, an end device sends a unicast transmission to its parent and allows the parent to route the data packet in its behalf.

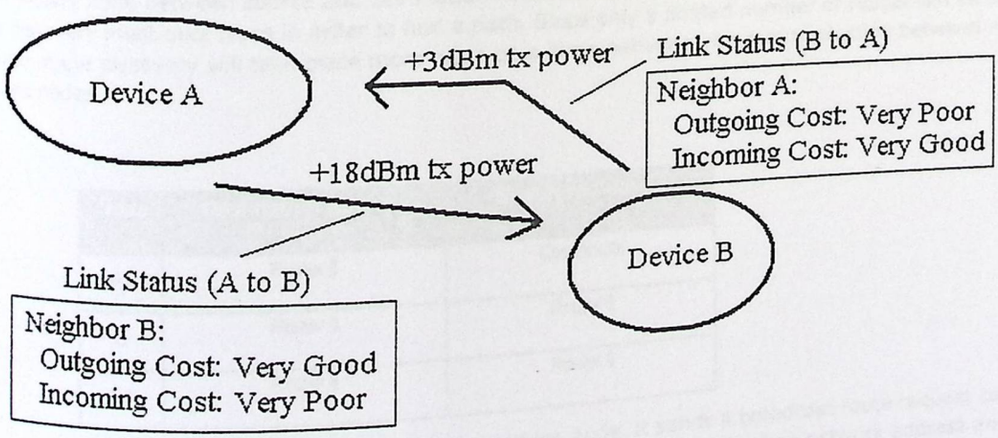
Note - A network cannot revert from Many-to-One routing to AODV routing without first doing a network reset (NR).

Link Status Transmission

Before discussing the various routing protocols, it is worth understanding the primary mechanism in ZigBee for establishing reliable bi-directional links. This mechanism is especially useful in networks that may have a mixture of devices with varying output power and/or receiver sensitivity levels.

Each coordinator or router device periodically sends a link status message. This message is sent as a 1-hop broadcast transmission, received only by one-hop neighbors. The link status message contains a list of neighboring devices and incoming and outgoing link qualities for each neighbor. Using these messages, neighboring devices can determine the quality of a bi-directional link with each neighbor and use that information to select a route that works well in both directions.

For example, consider a network of two neighboring devices that send periodic link status messages. Suppose that the output power of device A is +18dBm, and the output power of device B is +3dBm (considerably less than the output power of device A). The link status messages might indicate the following:



© 2012 Digi International, Inc.

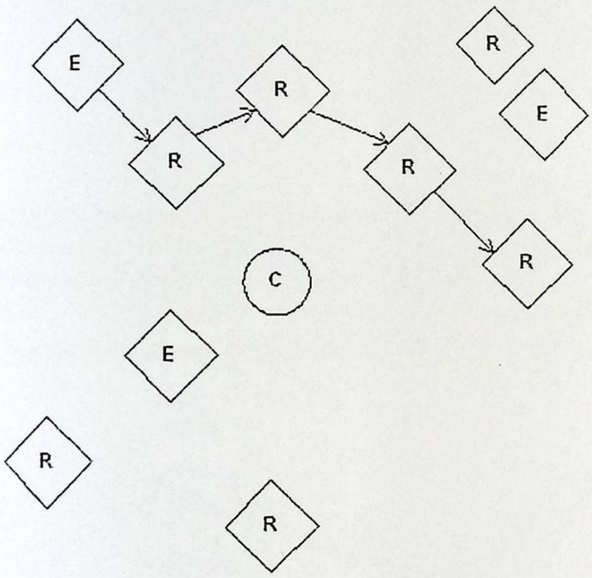
This mechanism enables devices A and B to recognize that the link is not reliable in both directions and select a different neighbor when establishing routes. (Such links are called asymmetric links, meaning the link quality is not similar in both directions.)

When a router or coordinator device powers on, it sends link status messages every couple seconds to attempt to discover link qualities with its neighbors quickly. After being powered on for some time, the link status messages are sent at a much slower rate (about every 3-4 times per minute).

AODV Mesh Routing

ZigBee employs mesh routing to establish a route between the source device and the destination. Mesh routing allows data packets to traverse multiple nodes (hops) in a network to route data from a source to a destination. Routers and coordinators can participate in establishing routes between source and destination devices using a process called route discovery. The Route discovery process is based on the AODV (Ad-hoc On-demand Distance Vector routing) protocol.

Sample Transmission Through a Mesh Network



AODV (Ad-hoc On-demand Distance Vector) Routing Algorithm

Routing under the AODV protocol is accomplished using tables in each node that store the next hop (intermediary node between source and destination nodes) for a destination node. If a next hop is not known, route discovery must take place in order to find a path. Since only a limited number of routes can be stored on a Router, route discovery will take place more often on a large network with communication between many different nodes.

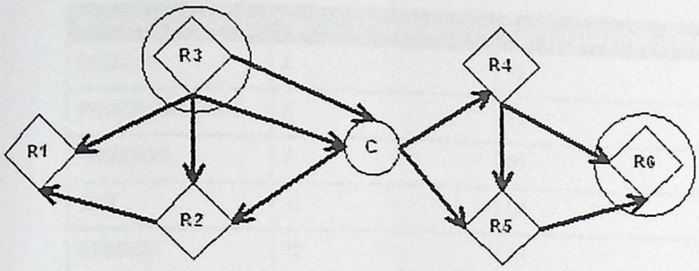
Node	Destination Address	Next Hop Address
R3	Router 6	Coordinator
C	Router 6	Router 5
R5	Router 6	Router 6

When a source node must discover a route to a destination node, it sends a broadcast route request command. The route request command contains the source network address, the destination network address and a path cost field (a metric for measuring route quality). As the route request command is propagated through the

© 2012 Digi International, Inc.

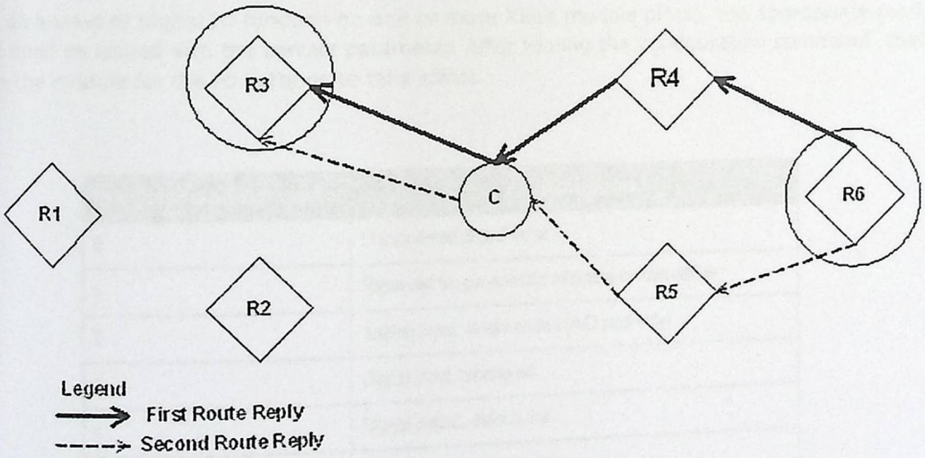
network (refer to the Broadcast Transmission), each node that re-broadcasts the message updates the path cost field and creates a temporary entry in its route discovery table.

Sample Route Request (Broadcast) Transmission Where R3 is Trying to Discover a Route to R6



When the destination node receives a route request, it compares the 'path cost' field against previously received route request commands. If the path cost stored in the route request is better than any previously received, the destination node will transmit a route reply packet to the node that originated the route request. Intermediate nodes receive and forward the route reply packet to the source node (the node that originated route request).

Sample Route Reply (Unicast) Where R6 Sends a Route Reply to R3.



Note: R6 could send multiple replies if it identifies a better route.

Retries and Acknowledgments

ZigBee includes acknowledgment packets at both the Mac and Application Support (APS) layers. When data is transmitted to a remote device, it may traverse multiple hops to reach the destination. As data is transmitted from one node to its neighbor, an acknowledgment packet (Ack) is transmitted in the opposite direction to indicate that the transmission was successfully received. If the Ack is not received, the transmitting device will retransmit the data, up to 4 times. This Ack is called the Mac layer acknowledgment.

In addition, the device that originated the transmission expects to receive an acknowledgment packet (Ack) from the destination device. This Ack will traverse the same path that the data traversed, but in the opposite

© 2012 Digi International, Inc.

8. XBee Analog and Digital IO Lines

XBee ZB firmware supports a number of analog and digital IO pins that are configured through software commands. Analog and digital IO lines can be set or queried. The following table lists the configurable IO pins and the corresponding configuration commands.

Module Pin Names	Module Pin Numbers	Configuration Command
CD/DIO12	4	P2
PWM0/RSSIM/DIO10	6	P0
PWM/DIO11	7	P1
DIO4	11	D4
CTS/DIO7	12	D7
ASSOC/DIO5	15	D5
RTS/DIO6	16	D6
AD3/DIO3	17	D3
AD2/DIO2	18	D2
AD1/DIO1	19	DI
AD0/DIO0	20	D0

IO Configuration

To enable an analog or digital IO function on one or more XBee module pin(s), the appropriate configuration command must be issued with the correct parameter. After issuing the configuration command, changes must be applied on the module for the IO settings to take effect.

Pin Command Parameter	Description
0	Unmonitored digital input
1	Reserved for pin-specific alternate functionalities
2	Analog input, single ended (A/D pins only)
3	Digital input, monitored
4	Digital output, default low
5	Digital output, default high
6-9	Alternate functionalities, where applicable

Pull-up resistors can be set for each digital input line using the PR command. The PR value updates the state of all pull-up resistors.

IO Sampling

The XBee ZB modules have the ability to monitor and sample the analog and digital IO lines. IO samples can be read locally or transmitted to a remote device to provide indication of the current IO line states. (Only API firmware devices can send remote IO sample data out their UART.)

There are three ways to obtain IO samples, either locally or remotely:

© 2012 Digi International, Inc.

- Queried Sampling
- Periodic Sampling
- Change Detection Sampling.

IO sample data is formatted as shown in the table below

Bytes	Name	Description
1	Sample Sets	Number of sample sets in the packet. (Always set to 1.)
2	Digital Channel Mask	Indicates which digital IO lines have sampling enabled. Each bit corresponds to one digital IO line on the module. • bit 0 = AD0/DIO0 • bit 1 = AD1/DIO1 • bit 2 = AD2/DIO2 • bit 3 = AD3/DIO3 • bit 4 = DIO4 • bit 5 = ASSOC/DIO5 • bit 6 = RTS/DIO6 • bit 7 = CTS/GPIO7 • bit 8 = N/A • bit 9 = N/A • bit 10 = RSSI/DIO10 • bit 11 = PWM/DIO11 • bit 12 = CD/DIO12 For example, a digital channel mask of 0x002F means DIO0,1,2,3, and 5 are enabled as digital IO.
1	Analog Channel Mask	Indicates which lines have analog inputs enabled for sampling. Each bit in the analog channel mask corresponds to one analog input channel. • bit 0 = AD0/DIO0 • bit 1 = AD1/DIO1 • bit 2 = AD2/DIO2 • bit 3 = AD3/DIO3 • bit 7 = Supply Voltage
Variable	Sampled Data Set	A sample set consisting of 1 sample for each enabled ADC and/or DIO channel, which has voltage inputs of 1143.75 and 342.1875mV. If any digital IO lines are enabled, the first two bytes of the data set indicate the state of all enabled digital IO. Only digital channels that are enabled in the Digital Channel Mask bytes have any meaning in the sample set. If no digital IO are enabled on the device, these 2 bytes will be omitted. Following the digital IO data (if any), each enabled analog channel will return 2 bytes. The data starts with AIN0 and continues sequentially for each enabled analog input channel up to AIN3, and the supply voltage (if enabled) at the end.

The sampled data set will include 2 bytes of digital IO data only if one or more IO lines on the device are configured as digital IO. If no pins are configured as digital IO, these 2 bytes will be omitted.

The digital IO data is only relevant if the same bit is enabled in the digital IO mask.

Analog samples are returned as 10-bit values. The analog reading is scaled such that 0x0000 represents 0V, and 0x3FF = 1.2V. (The analog inputs on the module cannot read more than 1.2V.) Analog samples are returned in order starting with AIN0 and finishing with AIN3, and the supply voltage. Only enabled analog input channels return data as shown in the figure below.

To convert the A/D reading to mV, do the following:

AD(mV) = (A/D reading * 1200mV) / 1023

© 2012 Digi International, Inc.

The reading in the sample frame represents voltage inputs of 1143.75 and 342.1875mV for AD0 and AD1 respectively.

Queried Sampling

The IS command can be sent to a device locally, or to a remote device using the API remote command frame (see chapter 8 for details). When the IS command is sent, the receiving device samples all enabled digital IO and analog input channels and returns an IO sample. If IS is sent locally, the IO sample is sent out the UART. If the IS command was received as a remote command, the IO sample is sent over-the-air to the device that sent the IS command.

If the IS command is issued in AT firmware, the module returns a carriage return-delimited list containing the above-listed fields. The API firmware returns an AT command response packet with the IO data included in the command data portion of the response frame.

The following table shows an example of the fields in an IS response.

Example	Sample AT Response
0x01	[1 sample set]
0x0C0C	[Digital Inputs: DIO 2, 3, 10, 11 low]
0x03	[Analog Inputs: A/D 0, 1]
0x0408	[Digital input states: DIO 3, 10 high, DIO 2, 11 low]
0x03D0	[Analog input ADIO 0= 0x3D0]
0x0124	[Analog input ADIO 1=0x120]

Periodic IO Sampling

Periodic sampling allows an XBee/XBee-PRO module to take an IO sample and transmit it to a remote device at a periodic rate. The periodic sample rate is set by the IR command. If IR is set to 0, periodic sampling is disabled. For all other values of IR, data will be sampled after IR milliseconds have elapsed and transmitted to a remote device. The DH and DL commands determine the destination address of the IO samples. DH and DL can be set to 0 to transmit to the coordinator, or to the 64-bit address of the remote device (SH and SL). Only devices running API firmware can send IO data samples out their UART. Devices running AT firmware will discard received IO data samples.

A sleeping end device will transmit periodic IO samples at the IR rate until the ST timer expires and the device can resume sleeping.

Change Detection Sampling

Modules can be configured to transmit a data sample immediately whenever a monitored digital IO pin changes state. The IC command is a bitmask that can be used to set which digital IO lines should be monitored for a state change. If one or more bits in IC is set, an IO sample will be transmitted as soon as a state change is observed in one of the monitored digital IO lines. Change detection samples are transmitted to the 64-bit address specified by DH and DL.

RSSI PWM

The XBee module features an RSSI/PWM pin (pin 6) that, if enabled, will adjust the PWM output to indicate the signal strength of the last received packet. The P0 (P-zero) command is used to enable the RSSI pulse width modulation (PWM) output on the pin. If P0 is set to 1, the RSSI/PWM pin will output a pulse width modulated signal where the frequency is adjusted based on the received signal strength of the last packet. Otherwise, for all other P0 settings, the pin can be used for general purpose IO.

When a data packet is received, if P0 is set to enable the RSSI/PWM feature, the RSSI PWM output is adjusted based on the RSSI of the last packet. The RSSI/PWM output will be enabled for a time based on the RP command. Each time an RF packet is received, the RSSI/PWM output is adjusted based on the RSSI of the new packet, and the RSSI timer is reset. If the RSSI timer expires, the RSSI/PWM pin is driven low. RP is measured in 100ms units and defaults to a value of 40 (4 seconds).

The RSSI PWM runs at 12MHz and has 2400 total counts (200us period).

RSSI (in dBm) is converted to PWM counts using the following equation:

PWM counts = (41 * RSSI_Unsigned) - 5928

IO Examples

Example 1: Configure the following IO settings on the XBee.

Configure AD1/DIO1 as a digital input with pullup resistor enabled

Configure AD2/DIO2 as an analog input

Configure DIO4 as a digital output, driving high.

To configure AD1/DIO1 as an input, issue the ATD1 command with a parameter of 3 ("ATD13"). To enable pull-up resistors on the same pin, the PR command should be issued with bit 3 set (e.g. ATPR8, ATPR1FFF, etc.).

The ATD2 command should be issued with a parameter of 2 to enable the analog input ("ATD22"). Finally, DIO4 can be set as an output, driving high by issuing the ATD4 command with a parameter value of 5 ("ATD45").

After issuing these commands, changes must be applied before the module IO pins will be updated to the new states. The AC or CN commands can be issued to apply changes (e.g. ATAC).

Example 2: Calculate the PWM counts for a packet received with an RSSI of -84dBm.

RSSI = -84 = 0xAC = 172 decimal (unsigned)

PWM counts = (41 * 172) - 5928

PWM counts = 1124

With a total of 2400 counts, this yields an ON time of (1124 / 2400) = 46.8%

Example 3: Configure the RSSI/PWM pin to operate for 2 seconds after each received RF packet.

First, ensure the RSSI/PWM functionality is enabled by reading the P0 (P-zero) command. It should be set to 1 (default).

To configure the duration of the RSSI/PWM output, set the RP command. To achieve a 2 second PWM output, set RP to 0x14 (20 decimal, or 2 seconds) and apply changes (AC command).

After applying changes, all received RF data packets should set the RSSI timer for 2 seconds.

© 2012 Digi International, Inc.

RSSI (in dBm) is converted to PWM counts using the following equation:

PWM counts = (41 * RSSI_Unsigned) - 5928

IO Examples

Example 1: Configure the following IO settings on the XBee.

Configure AD1/DIO1 as a digital input with pullup resistor enabled

Configure AD2/DIO2 as an analog input

Configure DIO4 as a digital output, driving high.

To configure AD1/DIO1 as an input, issue the ATD1 command with a parameter of 3 ("ATD13"). To enable pull-up resistors on the same pin, the PR command should be issued with bit 3 set (e.g. ATPR8, ATPR1FFF, etc.).

The ATD2 command should be issued with a parameter of 2 to enable the analog input ("ATD22"). Finally, DIO4 can be set as an output, driving high by issuing the ATD4 command with a parameter value of 5 ("ATD45").

After issuing these commands, changes must be applied before the module IO pins will be updated to the new states. The AC or CN commands can be issued to apply changes (e.g. ATAC).

Example 2: Calculate the PWM counts for a packet received with an RSSI of -84dBm.

RSSI = -84 = 0xAC = 172 decimal (unsigned)

PWM counts = (41 * 172) - 5928

PWM counts = 1124

With a total of 2400 counts, this yields an ON time of (1124 / 2400) = 46.8%

Example 3: Configure the RSSI/PWM pin to operate for 2 seconds after each received RF packet.

First, ensure the RSSI/PWM functionality is enabled by reading the P0 (P-zero) command. It should be set to 1 (default).

To configure the duration of the RSSI/PWM output, set the RP command. To achieve a 2 second PWM output, set RP to 0x14 (20 decimal, or 2 seconds) and apply changes (AC command).

After applying changes, all received RF data packets should set the RSSI timer for 2 seconds.

© 2012 Digi International, Inc.

API Operation

As an alternative to Transparent Operation, API (Application Programming Interface) Operations are available. API operation requires that communication with the module be done through a structured interface (data is communicated in frames in a defined order). The API specifies how commands, command responses and module status messages are sent and received from the module using a UART Data Frame.

Please note that Digi may add new API frames to future versions of firmware, so please build into your software interface the ability to filter out additional API frames with unknown Frame Types.

API Frame Specifications

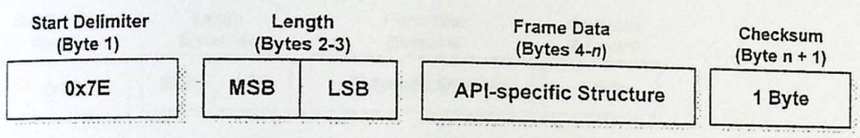
Two API modes are supported and both can be enabled using the AP (API Enable) command. Use the following AP parameter values to configure the module to operate in a particular mode:

- AP = 1: API Operation
- AP = 2: API Operation (with escaped characters)

API Operation (AP parameter = 1)

When this API mode is enabled (AP = 1), the UART data frame structure is defined as follows:

UART Data Frame Structure:



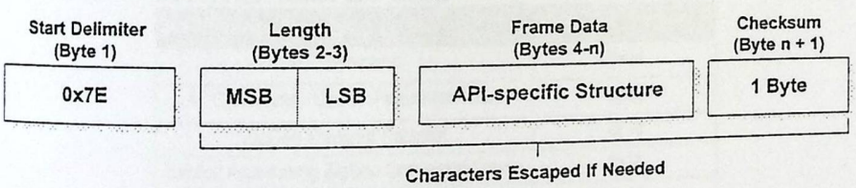
MSB = Most Significant Byte, LSB = Least Significant Byte

Any data received prior to the start delimiter is silently discarded. If the frame is not received correctly or if the checksum fails, the module will reply with a module status frame indicating the nature of the failure.

API Operation - with Escape Characters (AP parameter = 2)

When this API mode is enabled (AP = 2), the UART data frame structure is defined as follows:

UART Data Frame Structure - with escape control characters:



MSB = Most Significant Byte, LSB = Least Significant Byte

Escape characters. When sending or receiving a UART data frame, specific data values must be escaped (flagged) so they do not interfere with the data frame sequencing. To escape an interfering data byte, insert 0x7D and follow it with the byte to be escaped XOR'd with 0x20.

Data bytes that need to be escaped:

- 0x7E – Frame Delimiter
- 0x7D – Escape
- 0x11 – XON
- 0x13 – XOFF

Example - Raw UART Data Frame (before escaping interfering bytes):
0x7E 0x00 0x02 0x23 0x11 0xCB

0x11 needs to be escaped which results in the following frame:
0x7E 0x00 0x02 0x23 0x7D 0x31 0xCB

Note: In the above example, the length of the raw data (excluding the checksum) is 0x0002 and the checksum of the non-escaped data (excluding frame delimiter and length) is calculated as:
0xFF - (0x23 + 0x11) = (0xFF - 0x34) = 0xCB.

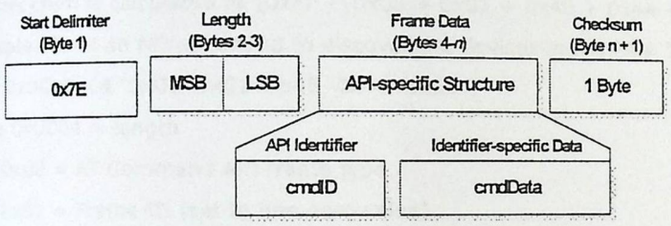
Length

The length field has a two-byte value that specifies the number of bytes that will be contained in the frame data field. It does not include the checksum field.

Frame Data

Frame data of the UART data frame forms an API-specific structure as follows:

UART Data Frame & API-specific Structure:



The cmdID frame (API-identifier) indicates which API messages will be contained in the cmdData frame (Identifier-specific data). Note that multi-byte values are sent big endian.The XBee modules support the following API frames:

API Frame Names and Values

API Frame Names	API ID
AT Command	0x08
AT Command - Queue Parameter Value	0x09
ZigBee Transmit Request	0x10
Explicit Addressing ZigBee Command Frame	0x11
Remote Command Request	0x17
Create Source Route	0x21
AT Command Response	0x88
Modem Status	0x8A
ZigBee Transmit Status	0x8B
ZigBee Receive Packet (AO=0)	0x90
ZigBee Explicit Rx Indicator (AO=1)	0x91
ZigBee IO Data Sample Rx Indicator	0x92
XBee Sensor Read Indicator (AO=0)	0x94
Node Identification Indicator (AO=0)	0x95
Remote Command Response	0x97
Over-the-Air Firmware Update Status	0xA0
Route Record Indicator	0xA1
Many-to-One Route Request Indicator	0xA3

©2012 Digi International, Inc.

Checksum

To test data integrity, a checksum is calculated and verified on non-escaped data.

To calculate: Not including frame delimiters and length, add all bytes keeping only the lowest 8 bits of the result and subtract the result from 0xFF.

To verify: Add all bytes (include checksum, but not the delimiter and length). If the checksum is correct, the sum will equal 0xFF.

API Examples

Example: Create an API AT command frame to configure an XBee to allow joining (set NJ to 0xFF). The frame should look like:

0x7E 0x00 0x05 0x08 0x01 0x4E 0x4A 0xFF 5F

Where 0x0005 = length

0x08 = AT Command API frame type

0x01 = Frame ID (set to non-zero value)

0x4E4A = AT Command ('NJ')

0xFF = value to set command to

0x5F = Checksum

The checksum is calculated as [0xFF - (0x08 + 0x01 + 0x4E + 0x4A + 0xFF)]

Example: Send an ND command to discover the devices in the PAN. The frame should look like:

0x7E 0x00 0x04 0x08 0x01 0x4E 0x44 0x64

Where 0x0004 = length

0x08 = AT Command API frame type

0x01 = Frame ID (set to non-zero value)

0x4E44 = AT command ('ND')

0x64 = Checksum

The checksum is calculated as [0xFF - (0x08 + 0x01 + 0x4E + 0x44)]

Example: Send a remote command to the coordinator to set AD1/DIO1 as a digital input (D1=3) and apply changes to force the IO update. The API remote command frame should look like:

0x7E 0x00 0x10 0x17 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xFF 0xFE 0x02 0x44 0x31 0x03 0x70

Where

0x10 = length (16 bytes excluding checksum)

0x17 = Remote Command API frame type

0x01 = Frame ID

0x0000000000000000 = Coordinator's address (can be replaced with coordinator's actual 64-bit address if known)

0xFFFE = 16- bit Destination Address

0x02 = Apply Changes (Remote Command Options)

0x4431 = AT command ('D1')

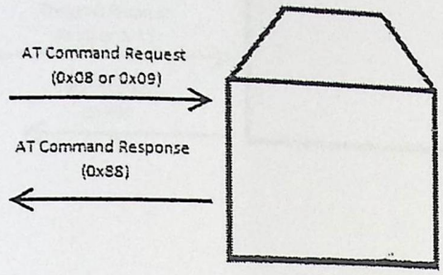
0x03 = Command Parameter (the parameter could also be sent as 0x0003 or 0x00000003)

0x70 = Checksum

API UART Exchanges

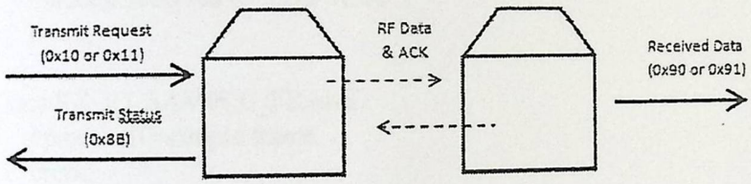
AT Commands

The following image shows the API frame exchange that takes place at the UART when sending an AT command request to read or set a module parameter. The response can be disabled by setting the frame ID to 0 in the request.



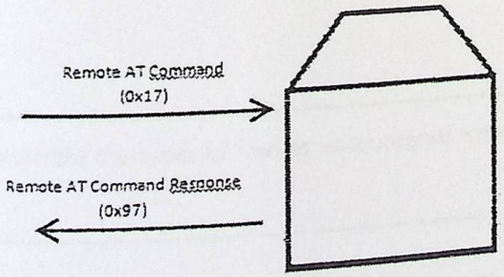
Transmitting and Receiving RF Data

The following image shows the API exchanges that take place at the UART when sending RF data to another device. The transmit status frame is always sent at the end of a data transmission unless the frame ID is set to 0 in the transmit request. If the packet cannot be delivered to the destination, the transmit status frame will indicate the cause of failure. The received data frame (0x90 or 0x91) is set by the AP command.



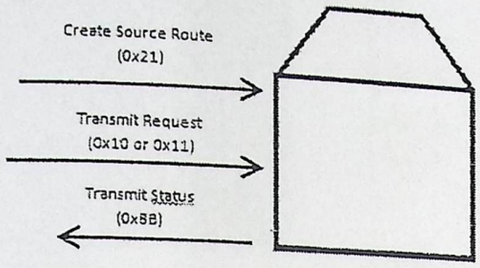
Remote AT Commands

The following image shows the API frame exchanges that take place at the UART when sending a remote AT command. A remote command response frame is not sent out the UART if the remote device does not receive the remote command.



Source Routing

The following image shows the API frame exchanges that take place at the UART when sending a source routed transmission.



Supporting the API

Applications that support the API should make provisions to deal with new API frames that may be introduced in future releases. For example, a section of code on a host microprocessor that handles received serial API frames (sent out the module's DOUT pin) might look like this:

```
void XBee_HandleRxAPIFrame(_apiFrameUnion *papiFrame){
  switch(papiFrame->api_id){
    case RX_RF_DATA_FRAME:
      //process received RF data frame
      break;

    case RX_IO_SAMPLE_FRAME:
      //process IO sample frame
      break;

    case NODE_IDENTIFICATION_FRAME:
      //process node identification frame
      break;

    default:
      //Discard any other API frame types that are not being used
      break;
  }
}
```

API Frames

The following sections illustrate the types of frames encountered while using the API.

AT Command

Frame Type: 0x08
Used to query or set module parameters on the local device. This API command applies changes after executing the command. (Changes made to module parameters take effect once changes are applied.) The API example below illustrates an API frame when modifying the NJ parameter value of the module

© 2012 Digi International, Inc.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x04	
	Frame-specific Data	Frame Type	3	0x08	
		Frame ID	4	0x52 (R)	Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement). If set to 0, no response is sent.
		AT Command	5	0x4E (N)	Command Name - Two ASCII characters that identify the AT Command.
			6	0x4A (J)	
		Parameter Value (optional)			If present, indicates the requested parameter value to set the given register. If no characters present, register is queried.
	Checksum		7	0x0D	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

The above example illustrates an AT command when querying an NJ value.

AT Command - Queue Parameter Value

Frame Type: 0x09

This API type allows module parameters to be queried or set. In contrast to the "AT Command" API type, new parameter values are queued and not applied until either the "AT Command" (0x08) API type or the AC (Apply Changes) command is issued. Register queries (reading parameter values) are returned immediately.

Example: Send a command to change the baud rate (BD) to 115200 baud, but don't apply changes yet. (Module will continue to operate at the previous baud rate until changes are applied.)

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x05	
	Frame-specific Data	Frame Type	3	0x09	
		Frame ID	4	0x01	Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement). If set to 0, no response is sent.
		AT Command	5	0x42 (B)	Command Name - Two ASCII characters that identify the AT Command.
			6	0x44 (D)	
		Parameter Value (ATBD7 = 115200 baud)	7	0x07	If present, indicates the requested parameter value to set the given register. If no characters present, register is queried.
	Checksum		8	0x68	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Note: In this example, the parameter could have been sent as a zero-padded 2-byte or 4-byte value.

ZigBee Transmit Request

Frame Type: 0x10

A Transmit Request API frame causes the module to send data as an RF packet to the specified destination.

The 64-bit destination address should be set to 0x000000000000FFFF for a broadcast transmission (to all devices). The coordinator can be addressed by either setting the 64-bit address to all 0x00s and the 16-bit address to 0xFFFE, OR by setting the 64-bit address to the coordinator's 64-bit address and the 16-bit address to 0x0000. For all other transmissions, setting the 16-bit address to the correct 16-bit address can help improve performance when transmitting to multiple destinations. If a 16-bit address is not known, this field should be set to 0xFFFE (unknown). The Transmit Status frame (0x8B) will indicate the discovered 16-bit address, if successful.

©2012 Digi International, Inc.

The broadcast radius can be set from 0 up to NH. If set to 0, the value of NH specifies the broadcast radius (recommended). This parameter is only used for broadcast transmissions.

The maximum number of payload bytes can be read with the NP command.

Note: if source routing is used, the RF payload will be reduced by two bytes per intermediate hop in the source route. This example shows if escaping is disabled (AP=1).

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x16	
	Frame-specific Data	Frame Type	3	0x10	
		Frame ID	4	0x01	Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement). If set to 0, no response is sent.
		64-bit Destination Address	MSB 5	0x00	Set to the 64-bit address of the destination device. The following addresses are also supported: 0x0000000000000000 - Reserved 64-bit address for the coordinator 0x000000000000FFFF - Broadcast address
			6	0x13	
			7	0xA2	
			8	0x00	
			9	0x40	
			10	0x0A	
			11	0x01	
			LSB 12	0x27	
		16-bit Destination Network Address	MSB 13	0xFF	Set to the 16-bit address of the destination device, if known. Set to 0xFFFE if the address is unknown, or if sending a broadcast.
			LSB 14	0xFE	
		Broadcast Radius	15	0x00	Sets maximum number of hops a broadcast transmission can occur. If set to 0, the broadcast radius will be set to the maximum hops value.
		Options	16	0x00	Bitfield of supported transmission options. Supported values include the following: 0x01 - Disable retries and route repair 0x20 - Enable APS encryption (if EE=1) 0x40 - Use the extended transmission timeout Enabling APS encryption presumes the source and destination have been authenticated. I also decreases the maximum number of RF payload bytes by 4 (below the value reported by NP). The extended transmission timeout is needed when addressing sleeping end devices.It also increases the retry interval between retries to compensate for end device polling.See Chapter 4, Transmission Timeouts, Extended Timeout for a description. Unused bits must be set to 0.
		RF Data	17	0x54	Data that is sent to the destination device
			18	0x78	
			19	0x44	
			20	0x61	
			21	0x74	
			22	0x61	
			23	0x30	
			24	0x41	
	Checksum		25	0x13	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: The example above shows how to send a transmission to a module where escaping is disabled (AP=1) with destination address 0x0013A200 40014011, payload "TxData1B". If escaping is enabled (AP=2), the frame should look like:

0x7E 0x00 0x16 0x10 0x01 0x00 0x7D 0x33 0xA2 0x00 0x40 0x0A 0x01 0x27
0xFF 0xFE 0x00 0x00 0x54 0x78 0x44 0x61 0x74 0x61 0x30 0x41 0x7D 0x33

The checksum is calculated (on all non-escaped bytes) as [0xFF - (sum of all bytes from API frame type through data payload)].

Example: Send a transmission to the coordinator without specifying the coordinator's 64-bit address. The API transmit request frame should look like:

0x7E 0x00 0x16 0x10 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xFF 0xFE 0x00 0x00 0x54 0x78 032 0x43 0x6F 0x6F 0x72 0x64 0xFC

Where 0x16 = length (22 bytes excluding checksum)

0x10 = ZigBee Transmit Request API frame type

0x01 = Frame ID (set to non-zero value)

0x0000000000000000 = Coordinator's address (can be replaced with coordinator's actual 64-bit address if known

0xFFFE = 16-bit Destination Address

0x00 = Broadcast radius

0x00 = Options

0x547832436F6F7264 = Data payload ("Tx2Coord")

0xFC = Checksum

Explicit Addressing ZigBee Command Frame

Frame Type: 0x11

Allows ZigBee application layer fields (endpoint and cluster ID) to be specified for a data transmission.

Similar to the ZigBee Transmit Request, but also requires ZigBee application layer addressing fields to be specified (endpoints, cluster ID, profile ID). An Explicit Addressing Request API frame causes the module to send data as an RF packet to the specified destination, using the specified source and destination endpoints, cluster ID, and profile ID.

The 64-bit destination address should be set to 0x000000000000FFFF for a broadcast transmission (to all devices). The coordinator can be addressed by either setting the 64-bit address to all 0x00s and the 16-bit address to 0xFFFE, OR by setting the 64-bit address to the coordinator's 64-bit address and the 16-bit address to 0x0000. For all other transmissions, setting the 16-bit address to the correct 16-bit address can help improve performance when transmitting to multiple destinations. If a 16-bit address is not known, this field should be set to 0xFFFE (unknown). The Transmit Status frame (0x8B) will indicate the discovered 16-bit address, if successful.

The broadcast radius can be set from 0 up to NH. If set to 0, the value of NH specifies the broadcast radius (recommended). This parameter is only used for broadcast transmissions.

The maximum number of payload bytes can be read with the NP command. Note: if source routing is used, the RF payload will be reduced by two bytes per intermediate hop in the source route.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x1A	
	Frame-specific Data	Frame Type	3	0x11	
		Frame ID	4	0x01	Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement). If set to 0, no response is sent.
		64-bit Destination Address	MSB 5	0x00	Set to the 64-bit address of the destination device. The following addresses are also supported: 0x0000000000000000 - Reserved 64-bit address for the coordinator 0x000000000000FFFF - Broadcast address
			6	0x00	
			7	0x00	
			8	0x00	
			9	0x00	
			10	0x00	
			11	0x00	
			12	0x00	
		16-bit Destination Network Address	MSB 13	0xFF	Set to the 16-bit address of the destination device, if known. Set to 0xFFFE if the address is unknown, or if sending a broadcast.
			LSB 14	0xFE	
		Source Endpoint	15	0xA0	Source endpoint for the transmission.
		Destination Endpoint	16	0xA1	Destination endpoint for the transmission.
		Cluster ID	17	0x15	Cluster ID used in the transmission
			18	0x54	
		Profile ID	19	0xC1	Profile ID used in the transmission
			20	0x05	
		Broadcast Radius	21	0x00	Sets the maximum number of hops a broadcast transmission can traverse. If set to 0, the transmission radius will be set to the network maximum hops value.
		Transmit Options	22	0x00	Bitfield of supported transmission options. Supported values include the following: 0x01 - Disable retries and route repair 0x20 - Enable APS encryption (if EE=1) 0x40 - Use the extended transmission timeout Enabling APS encryption presumes the source and destination have been authenticated. It also decreases the maximum number of RF payload bytes by 4 (below the value reported by NP). The extended transmission timeout is needed when addressing sleeping end devices. It also increases the retry interval between retries to compensate for end device polling. See Chapter 4, Transmission Timeouts, Extended Timeout for a description. Unused bits must be set to 0.
		Data Payload	23	0x54	Data that is sent to the destination device
			24	0x78	
			25	0x44	
			26	0x61	
			27	0x74	
			28	0x61	
	Checksum		29	0x3A	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

© Digi International, Inc.

Example: Send a data transmission to the coordinator (64-bit address of 0x00s) using a source endpoint of 0xA0, destination endpoint 0xA1, cluster ID =0x1554, and profile ID 0xC105. Payload will be "TxData".

© Digi International, Inc.

Example: Send a data transmission to the coordinator (64-bit address of 0x00s) using a source endpoint of 0xA0, destination endpoint 0xA1, cluster ID =0x1554, and profile ID 0xC105. Payload will be "TxData".

© Digi International, Inc.

Remote AT Command Request

Frame Type: 0x17

Used to query or set module parameters on a remote device. For parameter changes on the remote device to take effect, changes must be applied, either by setting the apply changes options bit, or by sending an AC command to the remote.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x10	
	Frame-specific Data	Frame Type	3	0x17	
		Frame ID	4	0x01	Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement). If set to 0, no response is sent.
		64-bit Destination Address	MSB 5	0x00	Set to the 64-bit address of the destination device. The following addresses are also supported: 0x0000000000000000 - Reserved 64-bit address for the coordinator 0x000000000000FFFF - Broadcast address
			6	0x13	
			7	0xA2	
			8	0x00	
			9	0x40	
			10	0x40	
			11	0x11	
			LSB 12	0x22	
		16-bit Destination Network Address	MSB 13	0xFF	Set to the 16-bit address of the destination device, if known. Set to 0xFFFE if the address is unknown, or if sending a broadcast.
			LSB 14	0xFE	
		Remote Command Options	15	0x02 (apply changes)	Bitfield of supported transmission options Supported values include the following: 0x01 - Disable retries and route repair 0x02 - Apply changes. 0x20 - Enable APS encryption (if EE=1) 0x40 - Use the extended transmission timeout If apply changes is not enabled, then an AC command must be sent before changes will take effect. Enabling APS encryption presumes the source and destination have been authenticated. It also decreases the maximum number of RF payload bytes by 4 (below the value reported by NP). The extended transmission timeout is needed when addressing sleeping end devices. It increases the retry interval between retries to compensate for end device polling. See Chapter 4, Transmission Timeouts, Extended Timeout for a description. Unused bits must be set to 0.
		AT Command	16	0x42 (B)	Name of the command
			17	0x48 (H)	
		Command Parameter	18	0x01	If present, indicates the requested parameter value to set the given register. If no characters present, the register is queried.
	Checksum		19	0xF5	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: Send a remote command to change the broadcast hops register on a remote device to 1 (broadcasts go to 1-hop neighbors only), and apply changes so the new configuration value immediately takes effect. In this example, the 64-bit address of the remote is 0x0013A200 40401122, and the destination 16-bit address is unknown.

Create Source Route

Frame Type: 0x21

This frame creates a source route in the module. A source route specifies the complete route a packet should traverse to get from source to destination. Source routing should be used with many-to-one routing for best results.

Note: Both the 64-bit and 16-bit destination addresses are required when creating a source route. These are obtained when a Route Record Indicator (0xA1) frame is received.

Frame Fields		Offset	Example	Description
Start Delimiter		0	0x7E	
Length		MSB 1	0x00	Number of bytes between the length and the checksum
		LSB 2	0x14	
Frame-specific Data	Frame Type	3	0x21	
	Frame ID	4	0x00	The Frame ID should always be set to 0.
	64-bit Destination Address	MSB 5	0x00	Set to the 64-bit address of the destination device. The following addresses are also supported: 0x0000000000000000 - Reserved 64-bit address for the coordinator 0x000000000000FFFF - Broadcast address
		6	0x13	
		7	0xA2	
		8	0x00	
		9	0x40	
		10	0x40	
		11	0x11	
		LSB 12	0x22	
	16-bit Destination Network Address	MSB 13	0x33	Set to the 16-bit address of the destination device, if known. Set to 0xFFFE if the address is unknown, or if sending a broadcast.
		LSB 14	0x44	
	Route Command Options	15	0x00	Set to 0.
	Number of Addresses	16	0x03	The number of addresses in the source route (excluding source and destination).
	Address 1	17	0xEE	(neighbor of destination)
		18	0xFF	
	Address 2 (closer hop	19	0xCC	Address of intermediate hop
		20	0xDD	
	Address 3	21	0xAA	(neighbor of source)
		22	0xBB	
Checksum		23	0x01	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: Intermediate hop addresses must be ordered starting with the neighbor of the destination, and working closer to the source. For example, suppose a route is found between A and E as shown below.

A'B'C'D'E

If device E has the 64-bit and 16-bit addresses of 0x0013A200 40401122 and 0x3344, and if devices B, C, and D have the following 16-bit addresses:

B = 0xAABB

C = 0xCCDD

D = 0xEEFF

The example above shows how to send the Create Source Route frame to establish a source route between A and E.

AT Command Response

Frame Type: 0x88

In response to an AT Command message, the module will send an AT Command Response message. Some commands will send back multiple frames (for example, the ND (Node Discover) command).

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x05	
	Frame-specific Data	Frame Type	3	0x88	
		Frame ID	4	0x01	Identifies the UART data frame being reported. Note: If Frame ID = 0 in AT Command Mode, no AT Command Response will be given.
		AT Command	5	'B' = 0x42	Command Name - Two ASCII characters that identify the AT Command.
			6	'D' = 0x44	
		Command Status	7	0x00	0 = OK 1 = ERROR 2 = Invalid Command 3 = Invalid Parameter 4 = Tx Failure
		Command Data			Register data in binary format. If the register was set, then this field is not returned, as in this example.
	Checksum		8	0xF0	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: Suppose the BD parameter is changed on the local device with a frame ID of 0x01. If successful (parameter was valid), the above response would be received.

Modem Status

Frame Type: (0x8A)

RF module status messages are sent from the module in response to specific conditions.

Example: The following API frame is returned when an API coordinator forms a network.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x02	
	Frame-specific Data	Frame Type	3	0x8A	
		Status	4	0x06	0 = Hardware reset 1 = Watchdog timer reset 2 =Joined network (routers and end devices) 3 =Disassociated 6 =Coordinator started 7 = Network security key was updated 0x0D = Voltage supply limit exceeded (PRO S2B only) 0x11 = Modem configuration changed while join in progress 0x80+ = stack error
	Checksum		5	0x6F	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Note: New modem status codes may be added in future firmware releases.

T Command Response

Frame Type: 0x88

In response to an AT Command message, the module will send an AT Command Response message. Some commands will send back multiple frames (for example, the ND (Node Discover) command).

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x05	
	Frame-specific Data	Frame Type	3	0x88	
		Frame ID	4	0x01	Identifies the UART data frame being reported. Note: If Frame ID = 0 in AT Command Mode, no AT Command Response will be given.
		AT Command	5	'B' = 0x42	Command Name - Two ASCII characters that identify the AT Command.
			6	'D' = 0x44	
		Command Status	7	0x00	0 = OK 1 = ERROR 2 = Invalid Command 3 = Invalid Parameter 4 = Tx Failure
		Command Data			Register data in binary format. If the register was set, then this field is not returned, as in this example.
	Checksum		8	0xF0	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: Suppose the BD parameter is changed on the local device with a frame ID of 0x01. If successful (parameter was valid), the above response would be received.

Modem Status

Frame Type: (0x8A)

RF module status messages are sent from the module in response to specific conditions.

Example: The following API frame is returned when an API coordinator forms a network.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x02	
	Frame-specific Data	Frame Type	3	0x8A	
		Status	4	0x06	0 = Hardware reset 1 = Watchdog timer reset 2 =Joined network (routers and end devices) 3 =Disassociated 6 =Coordinator started 7 = Network security key was updated 0x0D = Voltage supply limit exceeded (PRO S2B only) 0x11 = Modem configuration changed while join in progress 0x80+ = stack error
	Checksum		5	0x6F	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Note: New modem status codes may be added in future firmware releases.

API Packet	Length	1	0x00	Number of bytes between the length and the checksum
		2	0x8B	
	Frame Type	3	0x90	
	Frame ID	4	0x01	Identifies the UART data frame being reported. Note: If Frame ID = 0 in AT Command Mode, no AT Command Response will be given.
	16-bit address of destination	5	0x7D	16-bit Network Address the packet was delivered to (if success). If not success, this address matches the Destination Network Address that was provided in the Transmit Request Frame.
		6	0x84	
	Transmit Retry Count	7	0x00	The number of application transmission retries that took place.
	Frame-specific Data			
	Delivery Status	8	0x00	0x00 = Success 0x01 = MAC ACK Failure 0x02 = CCA Failure 0x15 = Invalid destination endpoint 0x21 = Network ACK Failure 0x22 = Not Joined to Network 0x23 = Self-addressed 0x24 = Address Not Found 0x25 = Route Not Found 0x26 = Broadcast source failed to hear a neighbor relay the message 0x2B = Invalid binding table index 0x2C = Resource error lack of free buffers, timers, etc. 0x2D = Attempted broadcast with APS transmission 0x2E = Attempted unicast with APS transmission, but EE=0 0x32 = Resource error lack of free buffers, timers, etc. 0x74 = Data payload too large 0x75 = Indirect message unrequested
	Discovery Status	9	0x01	0x00 = No Discovery Overhead 0x01 = Address Discovery 0x02 = Route Discovery 0x03 = Address and Route 0x40 = Extended Timeout Discovery
	Checksum	10	0x71	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: Suppose a unicast data transmission was sent to a destination device with a 16-bit address of 0x7D84. (The transmission could have been sent with the 16-bit address set to 0x7D84 or 0xFFFE.)

012 Digi International, Inc.

ZigBee Transmit Status

Frame Type: 0x8B

When a TX Request is completed, the module sends a TX Status message. This message will indicate if the packet was transmitted successfully or if there was a failure.

Start Delimiter	1	0x7E
Length	2	0x0007
Frame Type	3	0x8B
Frame ID	4	0x01
16-bit address of destination	5	0xFFFE
Transmit Retry Count	7	0x00
Delivery Status	8	0x00
Discovery Status	9	0x01
Checksum	10	0xFF

Sample frame: unicast data transmission was sent to a destination device with a 16-bit address of 0xFFFE. The transmission could have been sent with the 16-bit address set to 0x7D84.

ZigBee Receive Packet

Frame Type: (0x90)

When the module receives an RF packet, it is sent out the UART using this message type.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x11	
	Frame-specific Data	Frame Type	3	0x90	
		64-bit Source Address	MSB 4	0x00	64-bit address of sender. Set to 0xFFFFFFFFFFFFFFFF (unknown 64-bit address) if the sender's 64-bit address is unknown.
			5	0x13	
			6	0xA2	
			7	0x00	
			8	0x40	
			9	0x52	
			10	0x2B	
			LSB 11	0xAA	
		16-bit Source Network Address	MSB 12	0x7D	16-bit address of sender
			LSB 13	0x84	
		Receive Options	14	0x01	0x01 - Packet Acknowledged 0x02 - Packet was a broadcast packet 0x20 - Packet encrypted with APS encryption 0x40 - Packet was sent from an end device (if known) Note: Option values can be combined. For example, a 0x40 and a 0x01 will show as a 0x41. Other possible values 0x21, 0x22, 0x41, 0x42, 0x60, 0x61, 0x62.
		Received Data	15	0x52	Received RF data
			16	0x78	
			17	0x44	
			18	0x61	
			19	0x74	
			20	0x61	
	Checksum		21	0x0D	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: Suppose a device with a 64-bit address of 0x0013A200 40522BAA, and 16-bit address 0x7D84 sends a unicast data transmission to a remote device with payload "RxData". If AO=0 on the receiving device, it would send the above example frame out its UART.

ZigBee Explicit Rx Indicator

Frame Type:0x91

When the modem receives a ZigBee RF packet it is sent out the UART using this message type (when AO=1).

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x18	
	Frame-specific Data	Frame Type	3	0x91	
		64-bit Source Address	MSB 4	0x00	64-bit address of sender. Set to 0xFFFFFFFFFFFFFFFF (unknown 64-bit address) if the sender's 64-bit address is unknown.
			5	0x13	
			6	0xA2	
			7	0x00	
			8	0x40	
			9	0x52	
			10	0x2B	
			LSB 11	0xAA	
		16-bit Source Network Address	MSB 12	0x7D	16-bit address of sender.
			LSB 13	0x84	
		Source Endpoint	14	0xE0	Endpoint of the source that initiated the transmission
		Destination Endpoint	15	0xE0	Endpoint of the destination the message is addressed to.
		Cluster ID	16	0x22	Cluster ID the packet was addressed to.
			17	0x11	
		Profile ID	18	0xC1	Profile ID the packet was addressed to.
			19	0x05	
		Receive Options	20	0x02	0x01 – Packet Acknowledged 0x02 – Packet was a broadcast packet 0x20 - Packet encrypted with APS encryption 0x40 - Packet was sent from an end device (if known)
		Received Data	21	0x52	Received RF data
			22	0x78	
			23	0x44	
			24	0x61	
			25	0x74	
			26	0x61	
	Checksum		27	0x52	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: Suppose a device with a 64-bit address of 0x0013A200 40522BAA, and 16-bit address 0x7D84 sends a broadcast data transmission to a remote device with payload "RxData". Suppose the transmission was sent with source and destination endpoints of 0xE0, cluster ID=0x2211, and profile ID=0xC105. If AO=1 on the receiving device, it would send the above frame out its UART.

ZigBee IO Data Sample Rx Indicator

Frame Type: 0x92

When the module receives an IO sample frame from a remote device, it sends the sample out the UART using this frame type (when AO=0). Only modules running API firmware will send IO samples out the UART.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x14	
	Frame-specific Data	Frame Type	3	0x92	
		64-bit Source Address	MSB 4	0x00	64-bit address of sender
			5	0x13	
			6	0xA2	
			7	0x00	
			8	0x40	
			9	0x52	
			10	0x2B	
			LSB 11	0xAA	
		16-bit Source Network Address	MSB 12	0x7D	16-bit address of sender.
			LSB 13	0x84	
		Receive Options	14	0x01	0x01 - Packet Acknowledged 0x02 - Packet was a broadcast packet
		Number of Samples	15	0x01	Number of sample sets included in the payload. (Always set to 1)
		Digital Channel Mask*	16	0x00	Bitmask field that indicates which digital IO lines on the remote have sampling enabled (if any).
			17	0x1C	
		Analog Channel Mask**	18	0x02	Bitmask field that indicates which analog IO lines on the remote have sampling enabled (if any).
		Digital Samples (if included)	19	0x00	If the sample set includes any digital IO lines (Digital Channel Mask > 0), these two bytes contain samples for all enabled digital IO lines. DIO lines that do not have sampling enabled return 0. Bits in these 2 bytes map the same as they do in the Digital Channels Mask field.
			20	0x14	
		Analog Sample	21	0x02	If the sample set includes any analog input lines (Analog Channel Mask > 0), each enabled analog input returns a 2-byte value indicating the A/D measurement of that input. Analog samples are ordered sequentially from AD0/DIO0 to AD3/DIO3, to the supply voltage.
			22	0x25	
	Checksum		23	0xF5	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

*

N/A	N/A	N/A	CD/DIO 12	PWM/DIO11	RSSI/DIO10	N/A	N/A
CTS/DIO7	RTS/DIO6	ASSOC DIO5	DIO4	AD3/DIO3	AD2/DIO2	AD1/DIO1	AD0/DIO0

**

Supply Voltage	N/A	N/A	N/A	AD3	AD2	AD1	AD0

Example: Suppose an IO sample is received with analog and digital IO, from a remote with a 64-bit address of 0x0013A200 40522BAA and a 16-bit address of 0x7D84. If pin AD1/DIO1 is enabled as an analog input, AD2/DIO2 and DIO4 are enabled as a digital inputs (currently high), and AD3/DIO3 is enabled as a digital output (low) the IO sample is shown in the API example in the table above.

XBee Sensor Read Indicator

Frame Type: 0x94

When the module receives a sensor sample (from a Digi 1-wire sensor adapter), it is sent out the UART using this message type (when AO=0).

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x17	
		Frame Type	3	0x94	
			MSB 4	0x00	64-bit address of sender
			5	0x13	
		64-bit Source Address	6	0xA2	
			7	0x00	
			8	0x40	
			9	0x52	
			10	0x2B	
			LSB 11	0xAA	
		16-bit Source Network Address	MSB 12	0xDD	16-bit address of sender.
			LSB 13	0x6C	
	Frame-specific Data	Receive Options	14	0x01	0x01 - Packet Acknowledged 0x02 - Packet was a broadcast packet
		1-Wire Sensors	15	0x03	0x01 = A/D Sensor Read 0x02 = Temperature Sensor Read 0x60 = Water present (module CD pin low)
		A/D Values	16	0x00	Indicates a two-byte value for each of four A/D sensors (A, B, C, D) Set to 0xFFFFFFFFFFFFFFFF if no A/Ds are found.
			17	0x02	
			18	0x00	
			19	0xCE	
			20	0x00	
			21	0xEA	
			22	0x00	
			23	0x52	
		Temperature Read	24	0x01	Indicates the two-byte value read from a digital thermometer if present. Set to 0xFFFF if not found.
			25	0x6A	
	Checksum		26	0x8B	0xFF - the 0x8 bit sum of bytes from offset 3 to this byte.

Example: Suppose a 1-wire sensor sample is received from a device with a 64-bit address of 0x0013A200 40522BAA and a 16-bit address of 0xDD6C. If the sensor sample was taken from a 1-wire humidity sensor, the API frame could look like this (if AO=0):

For convenience, let's label the A/D and temperature readings as AD0, AD1, AD2, AD3, and T. Using the data in this example:

AD0 = 0x0002

AD1 = 0x00CE

AD2 = 0x00EA

AD3 = 0x0052

Example: Suppose an IO sample is received with analog and digital IO, from a remote with a 64-bit address of 0x0013A200 40522BAA and a 16-bit address of 0x7D84. If pin AD1/DIO1 is enabled as an analog input, AD2/DIO2 and DIO4 are enabled as a digital inputs (currently high), and AD3/DIO3 is enabled as a digital output (low) the IO sample is shown in the API example in the table above.

XBee Sensor Read Indicator

Frame Type: 0x94

When the module receives a sensor sample (from a Digi 1-wire sensor adapter), it is sent out the UART using this message type (when AO=0).

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x17	
		Frame Type	3	0x94	
		64-bit Source Address	MSB 4	0x00	64-bit address of sender
			5	0x13	
			6	0xA2	
			7	0x00	
			8	0x40	
			9	0x52	
			10	0x2B	
			LSB 11	0xAA	
		16-bit Source Network Address	MSB 12	0xDD	16-bit address of sender.
			LSB 13	0x6C	
		Receive Options	14	0x01	0x01 - Packet Acknowledged 0x02 - Packet was a broadcast packet
	Frame-specific Data	1-Wire Sensors	15	0x03	0x01 = A/D Sensor Read 0x02 = Temperature Sensor Read 0x60 = Water present (module CD pin low)
		A/D Values	16	0x00	Indicates a two-byte value for each of four A/D sensors (A, B, C, D) Set to 0xFFFFFFFFFFFFFFFF if no A/Ds are found.
			17	0x02	
			18	0x00	
			19	0xCE	
			20	0x00	
			21	0xEA	
			22	0x00	
			23	0x52	
		Temperature Read	24	0x01	Indicates the two-byte value read from a digital thermometer if present. Set to 0xFFFF if not found.
			25	0xEA	
	Checksum		26	0xBB	0xFF - the 8-bit sum of bytes from offset 3 to this byte

Example: Suppose a 1-wire sensor sample is received from a device with a 64-bit address of 0x0013A200 40522BAA and a 16-bit address of 0x7D84. In the sensor sample was taken from a 1-wire humidity sensor, the API frame could look like the of AD0, D1

For convenience, let's label the A/D and temperature readings as AD0, AD1, AD2, AD3, and T. Using the data in this example:

AD0 = 0x0002

AD1 = 0x00CE

AD2 = 0x00EA

AD3 = 0x0052

© 2012 Digi International, Inc.

T = 0x016A

To convert these to temperature and humidity values, the following equations should be used.

Temperature (°C) = (T / 16), for T < 2048

= - (T & 0x7FF) / 16, for T >= 2048

Vsupply = (AD2 * 5.1) / 255

Voutput = (AD3 * 5.1) / 255

Relative Humidity = ((Voutput / Vsupply) - 0.16) / (0.0062)

True Humidity = Relative Humidity / (1.0546 - (0.00216 * Temperature (°C)))

Looking at the sample data, we have:

Vsupply = (234 * 5.1 / 255) = 4.68

Voutput = (82 * 5.1 / 255) = 1.64

Temperature = (362 / 16) = 22.625°C

Relative H = (161.2903 * ((1.64/4.68) - 0.16)) = 161.2903 * (0.19043) = 30.71%

True H = (30.71 / (1.0546 - (0.00216 * 22.625))) = (30.71 / 1.00573) = 30.54%

©2012 Digi International, Inc.

Node Identification Indicator

Frame Type: 0x95

This frame is received when a module transmits a node identification message to identify itself (when AO=0). The data portion of this frame is similar to a network discovery response frame (see ND command).

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x20	
		Frame Type	3	0x95	
		64-bit Source Address	MSB 4	0x00	64-bit address of sender
			5	0x13	
			6	0xA2	
			7	0x00	
			8	0x40	
			9	0x52	
			10	0x2B	
			LSB 11	0xAA	
	Frame-specific Data	16-bit Source Network Address	MSB 12	0x7D	16-bit address of sender.
			LSB 13	0x84	
		Receive Options	14	0x02	0x01 - Packet Acknowledged 0x02 - Packet was a broadcast packet
		Source 16-bit address	15	0x7D	Set to the 16-bit network address of the remote. Set to 0xFFFE if unknown.
			16	0x84	
		64-bit Network address	17	0x00	Indicates the 64-bit address of the remote module that transmitted the node identification frame.
			18	0x13	
			19	0xA2	
			20	0x00	
			21	0x40	
			22	0x52	
			23	0x2B	
			24	0xAA	
		NI String	25	0x20	Node identifier string on the remote device. The NI-String is terminated with a NULL byte (0x00).
			26	0x00	
		Parent 16-bit address	27	0xFF	Indicates the 16-bit address of the remote's parent or 0xFFFE if the remote has no parent.
			28	0xFE	
		Device Type	29	0x01	0 = Coordinator 1 = Router 2 = End Device
		Source Event	30	0x01	1 = Frame sent by node identific ation pushbutton event (see D0 command) 2 = Frame sent after joining event occurred (see JN command). 3 = Frame sent after power cycle event occurred (see JN command).
		Digi Profile ID	31	0xC1	Set to Digi's application profile ID.
			32	0x05	
		Manufacturer ID	33	0x10	Set to Digi's Manufacturer ID.
			34	0x1E	
	Checksum		35	0x1B	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: If the commissioning push button is pressed on a remote router device with 64-bit address 0x0013A200 40522BAA, 16-bit address 0x7D84, and default NI string, the preceding

node identification indicator would be received. Please note that 00 03 00 00 appears before the checksum with the DD value only if ATNO & 0x01.

Remote Command Response

Frame Type: 0x97

If a module receives a remote command response RF data frame in response to a Remote AT Command Request, the module will send a Remote AT Command Response message out the UART. Some commands may send back multiple frames--for example, Node Discover (ND) command.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x13	
	Frame-specific Data	Frame Type	3	0x97	
		Frame ID	4	0x55	This is the same value passed in to the request.
		64-bit Source (remote) Address	MSB 5	0x00	The address of the remote radio returning this response.
			6	0x13	
			7	0xA2	
			8	0x00	
			9	0x40	
			10	0x52	
			11	0x2B	
			LSB 12	0xAA	
		16-bit Source (remote) Address	MSB 13	0x7D	Set to the 16-bit network address of the remote. Set to 0xFFFE if unknown.
			LSB 14	0x84	
		AT Commands	15	0x53	Name of the command
			16	0x4C	
		Command Status	17	0x00	0 = OK 1 = ERROR 2 = Invalid Command 3 = Invalid Parameter 4 = Remote Command Transmission Failed
		Command Data	18	0x40	Register data in binary format. If the register was set, then this field is not returned.
			19	0x52	
			20	0x2B	
			21	0xAA	
	Checksum		22	0xF0	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: If a remote command is sent to a remote device with 64-bit address 0x0013A200 40522BAA and 16-bit address 0x7D84 to query the SL command, and if the frame ID=0x55, the response is shown in the example API frame in the table above.

2 Digi International, Inc.

Over-the-Air Firmware Update Status

Frame Type: 0xA0

The Over-the-Air Firmware Update Status frame provides a status indication of a firmware update transmission attempt. If a query command (0x01 0x51) is sent to a target with a 64-bit address of 0x0013A200 40522BAA through an updater with 64-bit address 0x0013A200403E0750 and 16-bit address 0x0000, the following is the expected response.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x16	
	Frame-specific Data	Frame Type	3	0xA0	
		64-bit Source (remote) Address	MSB 4	0x00	The address of the remote radio returning this response.
			5	0x13	
			6	0xA2	
			7	0x00	
			8	0x40	
			9	0x3E	
			10	0x07	
			LSB 11	0x50	
		16-bit Destination Address	12	0x00	16-bit address of the updater device
			13	0x00	
		Receive Options	14	0x01	0x01 - Packet Acknowledged. 0x02 - Packet was a broadcast.
		Bootloader Message Type	15	0x52	0x06 - ACK 0x15 - NACK 0x40 - No Mac ACK 0x51 - Query (received if the bootloader is not active on the target) 0x52 - Query Response
		Block Number	16	0x00	Block number used in the update request. Set to 0 if not applicable.
		64-bit Target Address	17	0x00	64-bit Address of remote device that is being updated (target).
			18	0x13	
			19	0xA2	
			20	0x00	
			21	0x40	
			22	0x52	
			23	0x2B	
			24	0xAA	
	Checksum		25	0x66	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

If a query request returns a 0x15 (NACK) status, the target is likely waiting for a firmware update image. If no messages are sent to it for about 75 seconds, the target will timeout and accept new query messages.

If a query returns a 0x51 (QUERY) status, then the target's bootloader is not active and will not respond to query messages.

12 Digi International, Inc.

Route Record Indicator

Frame Type: 0xA1

The route record indicator is received whenever a device sends a ZigBee route record command. This is used with many-to-one routing to create source routes for devices in a network.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x13	
	Frame-specific Data	Frame Type	3	0xA1	
		64-bit Source Address	MSB 4	0x00	64-bit address of the device that initiated the route record.
			5	0x13	
			6	0xA2	
			7	0x00	
			8	0x40	
			9	0x40	
			10	0x11	
			LSB 11	0x22	
		Source (updater) 16-bit Address	12	0x33	16-bit address of the device that initiated the route record.
			13	0x44	
		Receive Options	14	0x01	0x01 - Packet Acknowledged. 0x02 - Packet was a broadcast.
		Number of Addresses	15	0x03	The number of addresses in the source route (excluding source and destination).
		Address 1	16	0xEE	(neighbor of destination)
			17	0xFF	
		Address 2 (closer hop	18	0xCC	Address of intermediate hop
			19	0xDD	
		Address n (neighbor of source)	20	0xAA	Two bytes per 16-bit address.
			21	0xBB	
	Checksum		22	0x80	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: Suppose device E sends a route record that traverses multiple hops en route to data collector device A as shown below.

A B C D E

If device E has the 64-bit and 16-bit addresses of 0x0013A200 40401122 and 0x3344, and if devices B, C, and D have the following 16-bit addresses:

B = 0xAABB

C = 0xCCDD

D = 0xEEFF

The data collector will send the above API frame out its UART.

012 Digi International, Inc.

Many-to-One Route Request Indicator

Frame Type: 0xA3

The many-to-one route request indicator frame is sent out the UART whenever a many-to-one route request is received

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x0C	
	Frame-specific Data	Frame Type	3	0xA3	
		64-bit Source Address	MSB 4	0x00	64-bit address of the device that sent the many-to-one route request
			5	0x13	
			6	0xA2	
			7	0x00	
			8	0x40	
			9	0x40	
			10	0x11	
			LSB 11	0x22	
		Source 16-bit Address	MSB 12	0x00	16-bit address of the device that initiated the many-to-one route request.
			LSB 13	0x00	
		Reserved	14	0x00	Set to 0.
	Checksum		15	0xF4	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: Suppose a device with a 64-bit address of 0x0013A200 40401122 and 16-bit address of 0x0000 sends a many-to-one route request. All remote routers with API firmware that receive the many-to-one broadcast would send the above example API frame out their UART.

ending ZigBee Device Objects (ZDO) Commands with the API

ZigBee Device Objects (ZDOs) are defined in the ZigBee Specification as part of the ZigBee Device Profile. These objects provide functionality to manage and map out the ZigBee network and to discover services on ZigBee devices. ZDOs are typically required when developing a ZigBee product that will interoperate in a public profile such as home automation or smart energy, or when communicating with ZigBee devices from other vendors. The ZDO can also be used to perform several management functions such as frequency agility (energy detect and channel changes - Mgmt Network Update Request), discovering routes (Mgmt Routing Request) and neighbors (Mgmt LQI Request), and managing device connectivity (Mgmt Leave and Permit Join Request).

The following table shows some of the more prominent ZDOs with their respective cluster identifier. Each ZDO command has a defined payload. See the "ZigBee Device Profile" section of the ZigBee Specification for details.

ZDO Command	Cluster ID
Network Address Request	0x0000
IEEE Address Request	0x0001
Node Descriptor Request	0x0002
Simple Descriptor Request	0x0004
Active Endpoints Request	0x0005
Match Descriptor Request	0x0006
Mgmt LQI Request	0x0031
Mgmt Routing Request	0x0032
Mgmt Leave Request	0x0034
Mgmt Permit Joining Request	0x0036
Mgmt Network Update Request	0x0038

The Explicit Transmit API frame (0x11) is used to send ZigBee Device Objects commands to devices in the network. Sending ZDO commands with the Explicit Transmit API frame requires some formatting of the data payload field.

When sending a ZDO command with the API, all multiple byte values in the ZDO command (API payload) (e.g. u16, u32, 64-bit addresses) must be sent in little endian byte order for the command to be executed correctly on a remote device.

For an API XBee to receive ZDO responses, the AO command must be set to 1 to enable the explicit receive API frame.

The following table shows how the Explicit API frame can be used to send an "Active Endpoints" request to discover the active endpoints on a device with a 16-bit address of 0x1234.

2 Digi International, Inc.

	Frame Fields		Offset	Example	Description
t Delimiter			0	0x7E	
gth			MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x17	
	Frame Type		3	0x11	
	Frame ID		4	0x01	Identifies the UART data frame for the host to correlate with a subsequent transmit status. If set to 0, no transmit status frame will be sent out the UART.
	64-bit Destination Address		MSB 5	0x00	64-bit address of the destination device (big endian byte order). For unicast transmissions, set to the 64-bit address of the destination device, or to 0x0000000000000000 to send a unicast to the coordinator. Set to 0x000000000000FFFF for broadcast.
			6	0x00	
			7	0x00	
			8	0x00	
			9	0x00	
			10	0x00	
			11	0xFF	
			12	0xFF	
	16-bit Destination Network Address		MSB 13	0xFF	16-bit address of the destination device (big endian byte order). Set to 0xFFFE for broadcast, or if the 16-bit address is unknown.
			LSB 14	0xFE	
me-specific Data	Source Endpoint		15	0x00	Set to 0x00 for ZDO transmissions (endpoint 0 is the ZDO endpoint).
	Destination Endpoint		16	0x00	Set to 0x00 for ZDO transmissions (endpoint 0 is the ZDO endpoint).
	Cluster ID		MSB 17	0x00	Set to the cluster ID that corresponds to the ZDO command being sent. 0x0005 = Active Endpoints Request
			LSB 18	0x05	
	Profile ID		MSB 19	0x00	Set to 0x0000 for ZDO transmissions (Profile ID 0x0000 is the ZigBee Device Profile that supports ZDOs).
			LSB 20	0x00	
	Broadcast Radius		21	0x00	Sets the maximum number of hops a broadcast transmission can traverse. If set to 0, the transmission radius will be set to the network maximum hops value.
	Transmit Options		22	0x00	All bits must be set to 0.
	Data Payload	Transaction Sequence Number	23	0x01	The required payload for a ZDO command. All multi-byte ZDO parameter values (u16, u32, 64-bit address) must be sent in little endian byte order. The Active Endpoints Request includes the following payload: [16-bit NwkAddrOfInterest] Note the 16-bit address in the API example (0x1234) is sent in little endian byte order (0x3412).
		ZDO Payload	24	0x34	
			25	0x12	
ecksum			26	0xA6	0xFF minus the 8 bit sum of bytes from offset 3 to this byte.

2 Digi International, Inc.

ending ZigBee Cluster Library (ZCL) Commands with the API

The ZigBee Cluster Library defines a set of attributes and commands (clusters) that can be supported in multiple ZigBee profiles. The ZCL commands are typically required when developing a ZigBee product that will interoperate in a public profile such as home automation or smart energy, or when communicating with ZigBee devices from other vendors. Applications that are not designed for a public profile or for interoperability applications can skip this section.

The following table shows some prominent clusters with their respective attributes and commands.

Cluster (Cluster ID)	Attributes (Attribute ID)	Cluster ID
Basic (0x0000)	Application Version (0x0001) Hardware Version (0x0003) Model Identifier (0x0005)	-Reset to defaults (0x00)
Identify (0x0003)	Identify Time (0x0000)	Identify (0x00) Identify Query (0x01)
Time (0x000A)	Time (0x0000) Time Status (0x0001) Time Zone (0x0002)	
Thermostat (0x0201)	Local Temperature (0x0000) Occupancy (0x0002)	-Setpoint raise / lower (0x00)

The ZCL defines a number of profile-wide commands that can be supported on any profile, also known as general commands. These commands include the following.

Command (Command ID)	Description
Read Attributes (0x00)	Used to read one or more attributes on a remote device.
Read Attributes Response (0x01)	Generated in response to a read attributes command.
Write Attributes (0x02)	Used to change one or more attributes on a remote device.
Write Attributes Response (0x04)	Sent in response to a write attributes command.
Configure Reporting (0x06)	Used to configure a device to automatically report on the values of one or more of its attributes.
Report Attributes (0x0A)	Used to report attributes when report conditions have been satisfied.
Discover Attributes (0x0C)	Used to discover the attribute identifiers on a remote device.
Discover Attributes Response (0x0D)	Sent in response to a discover attributes command.

The Explicit Transmit API frame (0x11) is used to send ZCL commands to devices in the network. Sending ZCL commands with the Explicit Transmit API frame requires some formatting of the data payload field.

When sending a ZCL command with the API, all multiple byte values in the ZCL command (API Payload) (e.g. u16, u32, 64-bit addresses) must be sent in little endian byte order for the command to be executed correctly on a remote device.

Note: When sending ZCL commands, the AO command should be set to 1 to enable the explicit receive API frame. This will provide indication of the source 64- and 16-bit addresses, cluster ID, profile ID, and endpoint information for each received packet. This information is required to properly decode received data.

12 Digi International, Inc.

The following table shows how the Explicit API frame can be used to read the hardware version attribute from a device with a 64-bit address of 0x0013A200 40401234 (unknown 16-bit address). This example uses arbitrary source and destination endpoints. Recall the hardware version attribute (attribute ID 0x0003) is part of the basic cluster (cluster ID 0x0000). The Read Attribute general command ID is 0x00.

Frame Fields			Offset	Example	Description
art Delimiter			0	0x7E	
ngth			MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x19	
	Frame Type		3	0x11	
	Frame ID		4	0x01	Identifies the UART data frame for the host to correlate with a subsequent transmit status. If set to 0, no transmit status frame will be sent out the UART.
	64-bit Destination Address		MSB 5	0x00	64-bit address of the destination device (big endian byte order). For unicast transmissions, set to the 64-bit address of the destination device, or to 0x0000000000000000 to send a unicast to the coordinator. Set to 0x000000000000FFFF for broadcast.
			6	0x13	
			7	0xA2	
			8	0x00	
			9	0x40	
			10	0x40	
			11	0x12	
			12	0x34	
	16-bit Destination Network Address		MSB 13	0xFF	16-bit address of the destination device (big endian byte order). Set to 0xFFFE for broadcast, or if the 16-bit address is unknown.
			LSB 14	0xFE	
	Source Endpoint		15	0x41	Set to the source endpoint on the sending device. (0x41 arbitrarily selected).
	Destination Endpoint		16	0x42	Set to the destination endpoint on the remote device. (0x42 arbitrarily selected)
ame-specific ata	Cluster ID		MSB 17	0x00	Set to the cluster ID that corresponds to the ZCL command being sent. 0x0000 = Basic Cluster
			LSB 18	0x00	
	Profile ID		MSB 19	0xD1	Set to the profile ID supported on the device. (0xD123 arbitrarily selected).
			LSB 20	0x23	
	Broadcast Radius		21	0x00	Sets the maximum number of hops a broadcast transmission can traverse. If set to 0, the transmission radius will be set to the network maximum hops value.
	Transmit Options		22	0x00	All bits must be set to 0.
	Data Payload	Frame Control	23	0x00	Bitfield that defines the command type and other relevant information in the ZCL command. See the ZCL specification for details.
	ZCL Frame Header	Transaction Sequence Number	24	0x01	A sequence number used to correlate a ZCL command with a ZCL response. (The hardware version response will include this byte as a sequence number in the response.) The value 0x01 was arbitrarily selected.
		Command ID	25	0x00	Since the frame control "frame type" bits are 00, this byte specifies a general command. Command ID 0x00 is a Read Attributes command.
	ZCL Payload	Attribute ID	26	0x03	The payload for a "Read Attributes" command is a list of Attribute Identifiers that are being read. Note the 16-bit Attribute ID (0x0003) is sent in little endian byte order (0x0300). All multi-byte ZCL header and payload values must be sent in little endian byte order.
			27	0x00	
Checksum			28	0xFA	0xFF minus the 8 bit sum of bytes from offset 3 to this byte.

2 Digi International, Inc.

In the above example, the Frame Control field (offset 23) was constructed as follows:

Name	Bits	Example Value Description
Frame Type	0-1	00 - Command acts across the entire profile
Manufacturer Specific	2	0 - The manufacturer code field is omitted from the ZCL Frame Header.
Direction	3	0 - The command is being sent from the client side to the server side.
Disable Default Response	4	0 - Default response not disabled
Reserved	5-7	Set to 0.

See the ZigBee Cluster Library specification for details.

Sending Public Profile Commands with the API

Commands in public profiles such as Smart Energy and Home Automation can be sent with the XBee API using the Explicit Transmit API frame (0x11). Sending public profile commands with the Explicit Transmit API frame requires some formatting of the data payload field. Most of the public profile commands fit into the ZigBee Cluster Library (ZCL) architecture as described in the previous section.

The following table shows how the Explicit API frame can be used to send a demand response and load control message (cluster ID 0x701) in the smart energy profile (profile ID 0x0109) in the revision 14 Smart Energy specification. The message will be a "Load Control Event" (command ID 0x00) and will be sent to a device with 64-bit address of 0x0013A200 40401234 with a 16-bit address of 0x5678. The event will start a load control event for water heaters and smart appliances, for a duration of 1 minute, starting immediately.

Note: When sending public profile commands, the AO command should be set to 1 to enable the explicit receive API frame. This will provide indication of the source 64- and 16-bit addresses, cluster ID, profile ID, and endpoint information for each received packet. This information is required to properly decode received data.

Frame Fields			Offset	Example	Description
Start Delimiter			0	0x7E	
Length			MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x19	
	Frame Type		3	0x11	
	Frame ID		4	0x01	Identifies the UART data frame for the host to correlate with a subsequent transmit status. If set to 0, no transmit status frame will be sent out the UART.
	64-bit Destination Address		MSB 5	0x00	64-bit address of the destination device (big endian byte order). For unicast transmissions, set to the 64-bit address of the destination device, or to 0x0000000000000000 to send a unicast to the coordinator. Set to 0x000000000000FFFF for broadcast.
			6	0x13	
			7	0xA2	
			8	0x00	
			9	0x40	
			10	0x40	
			11	0x12	
			12	0x34	
	16-bit Destination Network Address		MSB 13	0x56	16-bit address of the destination device (big endian byte order). Set to 0xFFFE for broadcast, or if the 16-bit address is unknown.
			LSB 14	0x78	
	Source Endpoint		15	0x41	Set to the source endpoint on the sending device. (0x41 arbitrarily selected).
Frame-specific Data	Destination Endpoint		16	0x42	Set to the destination endpoint on the remote device. (0x42 arbitrarily selected)
	Cluster ID		MSB 17	0x07	Set to the cluster ID that corresponds to the ZCL command being sent. 0x0701 = Demand response and load control cluster ID
			LSB 18	0x01	
	Profile ID		MSB 19	0x01	Set to the profile ID supported on the device. 0x0109 = Smart Energy profile ID.
			LSB 20	0x09	
	Broadcast Radius		21	0x00	Sets the maximum number of hops a broadcast transmission can traverse. If set to 0, the transmission radius will be set to the network maximum hops value.
	Transmit Options		22	0x00	All bits must be set to 0.
	Data Payload	Frame Control	23	0x09	Bitfield that defines the command type and other relevant information in the ZCL command. See the ZCL specification for details.
	ZCL Frame Header	Transaction Sequence Number	24	0x01	A sequence number used to correlate a ZCL command with a ZCL response. (The hardware version response will include this byte as a sequence number in the response.) The value 0x01 was arbitrarily selected.
			25	0x00	Since the frame control "frame type" bits are 01, this byte specifies a cluster-specific command. Command ID 0x00 in the Demand Response and Load Control cluster is a Load Control Event command. (See Smart Energy specification.)

Frame Fields			Offset	Example	Description
		Issuer Event ID	26	0x78	4-byte unique identifier. Note the 4-byte ID is sent in little endian byte order (0x78563412). The event ID in this example (0x12345678) was arbitrarily selected.
			27	0x56	
			28	0x34	
			29	0x12	
		Device Class	30	0x14	to apply the load control event. A bit value of 0x0014 enables smart appliances and water heaters. Note the 2-byte bit field value is sent in little endian byte order.
			31	0x00	
		Utility Enrollment Group	32	0x00	Used to identify sub-groups of devices in the device-class. 0x00 addresses all groups.
		Start Time	33	0x00	UTC timestamp representing when the event should start. A value of 0x00000000 indicates "now".
			34	0x00	
			35	0x00	
			36	0x00	
		Duration in Minutes	37	0x01	This 2-byte value must be sent in little endian byte order.
			38	0x00	
	ZCL Payload - Load Control Event Data	Criticality Level	39	0x04	Indicates the criticality level of the event. In this example, the level is "voluntary".
		Cooling Temperature	40	0xFF	Requested offset to apply to the normal cooling set point. A value of 0xFF indicates the temperature offset value is not used.
		Heating Temperature Offset	41	0xFF	Requested offset to apply to the normal heating set point. A value of 0xFF indicates the temperature offset value is not used.
		Cooling Temperature Set Point	42	0x00	Requested cooling set point in 0.01 degrees Celsius. A value of 0x8000 means the set point field is not used in this event. Note the 0x80000 is sent in little endian byte order.
			43	0x80	
		Heating Temperature Set Point	44	0x00	Requested heating set point in 0.01 degrees Celsius. A value of 0x8000 means the set point field is not used in this event. Note the 0x80000 is sent in little endian byte order.
			45	0x80	
		Average Load Adjustment Percentage	46	0x80	Maximum energy usage limit. A value of 0x80 indicates the field is not used.
		Duty Cycle	47	0xFF	Defines the maximum "On" duty cycle. A value of 0xFF indicates the duty cycle is not used in this event.
		Duty Cycle Event Control	48	0x00	A bitmap describing event options.
...hecksum			49	0x5B	0xFF minus the 8 bit sum of bytes from offset 3 to this byte.

In the above example, the Frame Control field (offset 23) was constructed as follows:

Name	Bits	Example Value Description
Frame Type	0-1	01 - Command is specific to a cluster
Manufacturer Specific	2	0 - The manufacturer code field is omitted from the ZCL Frame Header.
Direction	3	1 - The command is being sent from the server side to the client side.
Disable Default Response	4	0 - Default response not disabled
Reserved	5-7	Set to 0.

Frame Fields		Offset	Example	Description
	Issuer Event ID	26	0x78	4-byte unique identifier. Note the 4-byte ID is sent in little endian byte order (0x78563412). The event ID in this example (0x12345678) was arbitrarily selected.
		27	0x56	
		28	0x34	
		29	0x12	
	Device Class	30	0x14	to apply the load control event. A bit value of 0x0014 enables smart appliances and water heaters. Note the 2-byte bit field value is sent in little endian byte order.
		31	0x00	
	Utility Enrollment Group	32	0x00	Used to identify sub-groups of devices in the device-class. 0x00 addresses all groups.
	Start Time	33	0x00	UTC timestamp representing when the event should start. A value of 0x00000000 indicates "now".
		34	0x00	
		35	0x00	
		36	0x00	
	Duration in Minutes	37	0x01	This 2-byte value must be sent in little endian byte order.
		38	0x00	
ZCL Payload - Load Control Event Data	Criticality Level	39	0x04	Indicates the criticality level of the event. In this example, the level is "voluntary".
	Cooling Temperature	40	0xFF	Requested offset to apply to the normal cooling set point. A value of 0xFF indicates the temperature offset value is not used.
	Heating Temperature Offset	41	0xFF	Requested offset to apply to the normal heating set point. A value of 0xFF indicates the temperature offset value is not used.
	Cooling Temperature Set Point	42	0x00	Requested cooling set point in 0.01 degrees Celsius. A value of 0x8000 means the set point field is not used in this event. Note the 0x80000 is sent in little endian byte order.
		43	0x80	
	Heating Temperature Set Point	44	0x00	Requested heating set point in 0.01 degrees Celsius. A value of 0x8000 means the set point field is not used in this event. Note the 0x80000 is sent in little endian byte order.
		45	0x80	
	Average Load Adjustment Percentage	46	0x80	Maximum energy usage limit. A value of 0x80 indicates the field is not used.
	Duty Cycle	47	0xFF	Defines the maximum "On" duty cycle. A value of 0xFF indicates the duty cycle is not used in this event.
	Duty Cycle Event Control	48	0x00	A bitmap describing event options.
hecksum		49	0x5B	0xFF minus the 8 bit sum of bytes from offset 3 to this byte.

In the above example, the Frame Control field (offset 23) was constructed as follows:

Name	Bits	Example Value Description
Frame Type	0-1	01 - Command is specific to a cluster
Manufacturer Specific	2	0 - The manufacturer code field is omitted from the ZCL Frame Header.
Direction	3	1 - The command is being sent from the server side to the client side.
Disable Default Response	4	0 - Default response not disabled
Reserved	5-7	Set to 0.

Appendix B

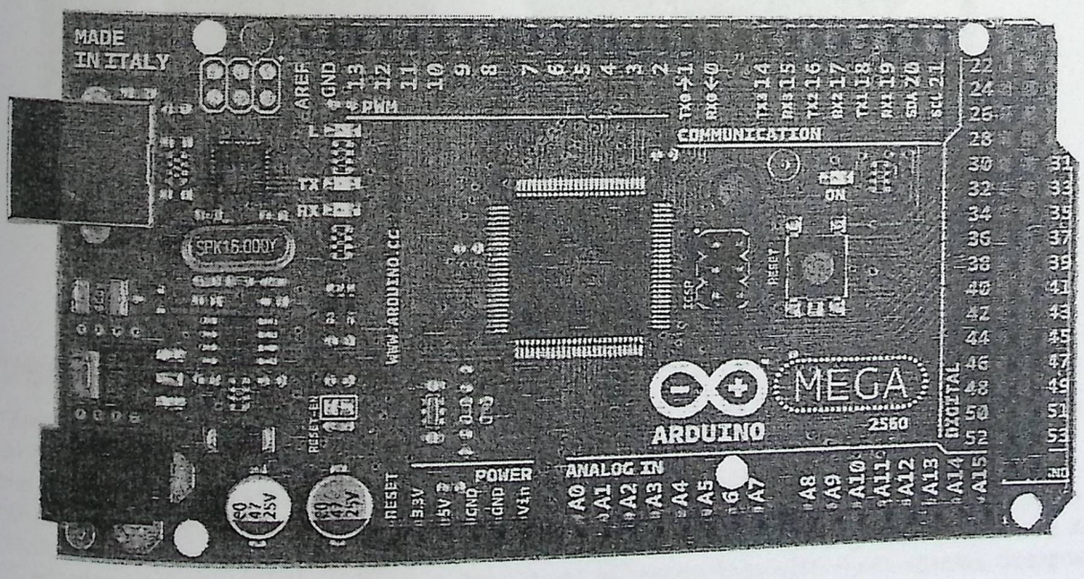
RobotShop

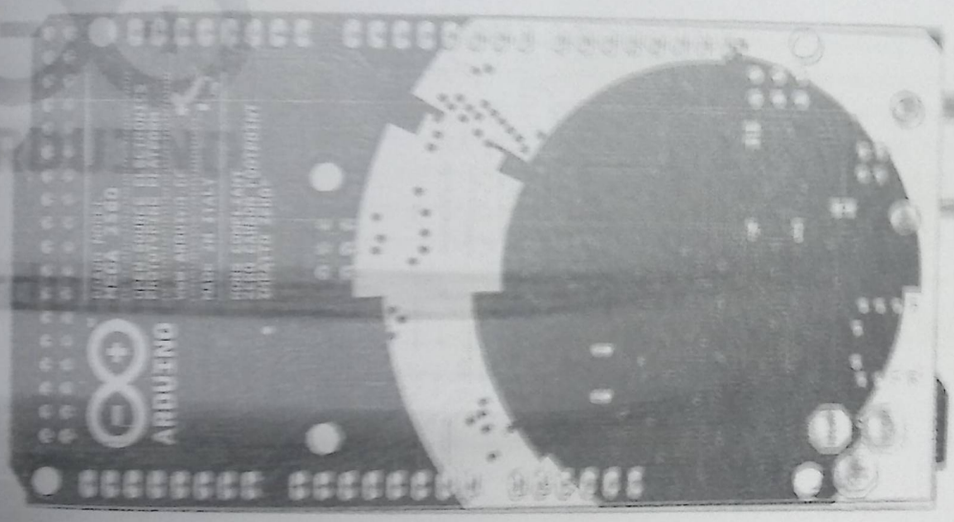
www.robotshop.com

La robotique à votre service! - Robotics at your service!

ARDUINO_

Arduino Mega 2560 Datasheet





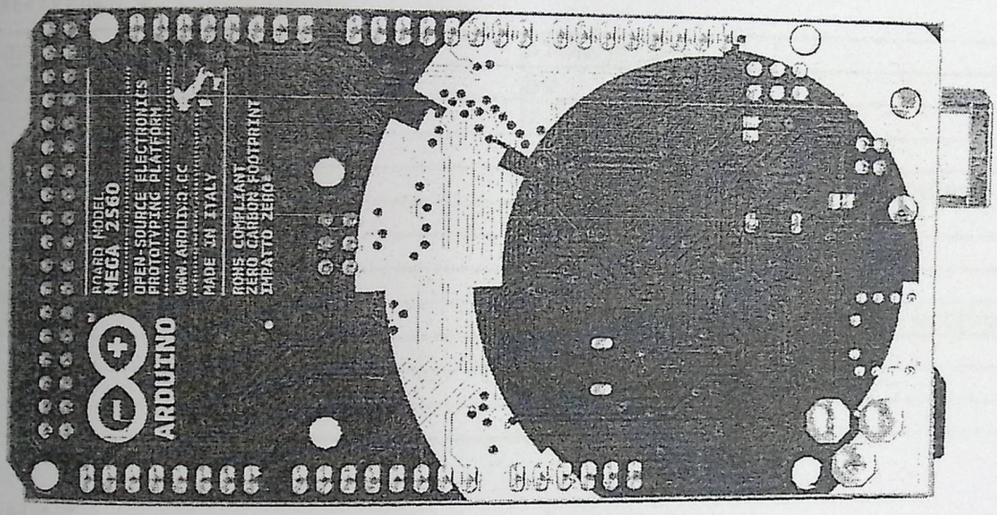
Overview

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560 (datasheet). It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila.

Schematic & Reference Design

EAGLE files: arduino-mega2560-reference-design.zip

RobotShop
www.robotshop.com
La robotique à votre service! - Robotics at your service!



Overview

he Arduino Mega 2560 is a microcontroller board based on the ATmega2560 (datasheet). t has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog nputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a ower jack, an ICSP header, and a reset button. It contains everything needed to support he microcontroller; simply connect it to a computer with a USB cable or power it with a AC-o-DC adapter or battery to get started. The Mega is compatible with most shields designed or the Arduino Duemilanove or Diecimila.

Schematic & Reference Design

AGLE files: arduino-mega2560-reference-design.zip

RobotShop
www.robotshop.com
La robotique à votre service! - Robotics at your service!

Schematic: arduino-mega2560-schematic.pdf

Summary

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 14 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

Power

he Arduino Mega can be powered via the USB connection or with an external power supply. he power source is selected automatically.

xternal (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or attery. The adapter can be connected by plugging a 2.1mm center-positive plug into the oard's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of e POWER connector.

he board can operate on an external supply of 6 to 20 volts. If supplied with less than /, however, the 5V pin may supply less than five volts and the board may be unstable. using more than 12V, the voltage regulator may overheat and damage the board. The commended range is 7 to 12 volts.

e Mega2560 differs from all preceding boards in that it does not use the FTDI USB-to-rial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial nverter.

RobotShop
www.robotshop.com
La robotique à votre service! - Robotics at your service!

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 54 digital pins on the Mega can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 0 to 13.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication using the SPI library. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Uno, Duemilanove and Diecimila.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH

RobotShop
www.robotshop.com
La robotique à votre service! - Robotics at your service!

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 54 digital pins on the Mega can be used as an input or output, using pinMode() digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 0 to 13.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication using the SPI library. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Uno, Duemilanove and Diecimila.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH

RobotShop
www.robotshop.com
La robotique à votre service! - Robotics at your service!

value, the LED is on, when the pin is LOW, it's off.

- **I2C: 20 (SDA) and 21 (SCL).** Support I2C (TWI) communication using the Wire library (documentation on the Wiring website). Note that these pins are not in the same location as the I2C pins on the Duemilanove or Diecimila.

The Mega2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Communication

The Arduino Mega2560 has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega2560 provides four hardware UARTs for TTL (5V) serial communication. An ATmega8U2 on the board channels one of these over USB and provides a virtual com port to software on the computer (Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega8U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).
A SoftwareSerial library allows for serial communication on any of the Mega2560's digital pins.
The ATmega2560 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation on the Wiring website for details. For SPI communication, use the SPI library.

Programming

The Arduino Mega can be programmed with the Arduino software (download). For details, see the reference and tutorials.
The ATmega2560 on the Arduino Mega comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It

RobotShop
www.robotshop.com
La robotique à votre service! - Robotics at your service!

communicates using the original STK500 protocol (reference, C header files).
You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.

Automatic (Software) Reset

Rather then requiring a physical press of the reset button before an upload, the Arduino Mega2560 is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATmega2560 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.
This setup has other implications. When the Mega2560 is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Mega2560. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.
The Mega2560 contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino Mega2560 has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics and Shield Compatibility

The maximum length and width of the Mega2560 PCB are 4 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

The Mega2560 is designed to be compatible with most shields designed for the Uno, Diecimila or Duemilanove. Digital pins 0 to 13 (and the adjacent AREF and GND pins), analog inputs 0 to 5, the power header, and ICSP header are all in equivalent locations. Further the main UART (serial port) is located on the same pins (0 and 1), as are external interrupts 0 and 1 (pins 2 and 3 respectively). SPI is available through the ICSP header on both the Mega2560 and Duemilanove / Diecimila. *Please note that I2C is not located on the same pins on the Mega (20 and 21) as the Duemilanove / Diecimila (analog inputs 4 and 5).*

Reference

[1]: Graduation project," Wireless Operated Parking System ",
Sh.Arafeh, Sh.Wazwaz, M.Al-juneidi, Palestine Polytechnic university 2011.

[2] Graduation project," Smart Parking System Using RFID Technology ",
A.Al-fouqha, M.Al-masri, AN-Najah National University, 2007.

[3] Graduation project," ZigBee-Based Monitoring System for People with Mental Disabilities ", D.Herbawi, S.Al-Tamimi, L.Sharabati, Y.Al-Qaisi, Palestine Polytechnic University, 2013.

[4]: [Online] arduino microcontroller product specification, feature, software

http://arduino.cc/en/Guide/Introduction

[5]: [Online] The latest news and product updates

http://www.android.com/about/

[6]: [Online] C Sharp (programming language)
http://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29

[7]: [Online] Java (programming language)
http://en.wikipedia.org/wiki/Java_%28programming_language%29

[8]: [Online] Tutorials about Eclipse, Java, Android and Web development
http://www.vogella.com/articles/AndroidSQLite/article.html

[9]: [Online] Difference between java and C#

http://www.differencebetween.info/difference-between-java-and-c-sharp

[10]: [Online]: Microsoft Office Access

https://en.wikipedia.org/wiki/Microsoft_Access

[11]: [Online]: MS SQL capable and used

http://www.host-shopper.com/what-is-ms-sql.html