การออกแบบและพัฒนาโนดศูนย์กลางและโนดเคลื่อนที่สำหรับโครงข่ายวัดระยะไกลด้วยเซนเซอร์

นายกี เล็ง

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2547

ISBN 974-53-1197-9

DESIGN AND DEVELOPMENT OF CENTRAL NODE AND MOBILE NODE FOR TELE-MEASUREMENT SENSOR NETWORK

Mr. KY-Leng

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Electrical Engineering
Department of Electrical Engineering
Faculty of Engineering
Chulalongkorn University
Academic year 2004

ISBN 974-53-1197-9

Thesis Title	DESIGN AND DEVELOPMENT OF CENTRAL NODE AND MOBILE NODE FOR TELE-MEASUREMENT SENSOR NETWORK
By	Mr. KY-Leng
Field of Study	Electrical Engineering
Thesis Advisor	Associate Professor Watit Benjapolakul, D. Eng

---

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

[illegible] ....Dean of the Faculty of Engineering
(Professor Direk Lavansiri, Ph. D.)

THESIS COMMITTEE

[illegible] .... Chairman
(Somboon Chongchaikit, D. Ing.)

[illegible] .... Thesis Advisor
(Associate Professor Watit Benjapolakul, D. Eng.)

[illegible] .... Member
(Chaiyachet Saivichit, Ph. D.)

[illegible] .... Member
(Assistant Professor Chaodit Aswakul, Ph. D.)

กี เล็ง : การออกแบบและพัฒนาโนดศูนย์กลางและโนดเคลื่อนที่สำหรับโครงข่ายวัดระยะไกลด้วยเซนเซอร์. (DESIGN AND DEVELOPMENT OF CENTRAL NODE AND MOBILE NODE FOR TELE-MEASUREMENT SENSOR NETWORK) อ. ที่ปรึกษา : รศ. ดร.วาทิต เบญจพลกุล, จำนวนหน้า 152 หน้า. ISBN 974-53-1197-9.

โครงข่ายสภาวะแวดล้อมแบบดั้งเดิมไม่สามารถป้อนข้อมูลแก่อุปกรณ์ควบคุม และไม่เป็นที่น่าสนใจสำหรับผู้ใช้งานทั่วไป ดังนั้น เราจึงเสนอโครงข่ายวัดด้วยเซนเซอร์ส่วนบุคคลเพื่อการเฝ้าสังเกตสภาวะแวดล้อมและระบบรักษาความปลอดภัยระยะไกลสำหรับบริษัท องค์กรและผู้ใช้ทั่วไป

ความสามารถในการเก็บรวบรวมข้อมูลระยะยาว ซึ่งถูกรวบรวมอยู่ในส่วนของงานวิจัยหลายๆ งานวิจัย โดยเฉพาะอย่างยิ่งวิศวกรรมสำรวจ เพื่อเก็บรวบรวมข้อมูลด้วยวิธีการที่ง่ายกว่าวิธีการแบบดั้งเดิม

ด้วยความสามารถด้านการป้อนกลับข้อมูลให้กับอุปกรณ์ควบคุมนี้ ทำให้ผู้ใช้งานทั่วไปสามารถเฝ้าสังเกตและควบคุมระบบได้ง่ายและมีความเชื่อถือได้มากยิ่งขึ้น

โครงข่ายที่เสนอนี้ ทำให้ผู้ใช้ทั่วไปสามารถเปลี่ยนรูปลักษณ์ การปรับระบบให้ทันกาลได้รวดเร็ว และง่ายขึ้น ด้วยการใช้เทคนิคมอดุลาร์และความสามารถในการตั้งรูปลักษณ์ขึ้นใหม่จากศูนย์กลาง สามารถใช้งานได้ทั้งภายในและภายนอกโครงข่าย

ด้วยการเลือกเทคโนโลยีที่มีความเหมาะสมจากไมโครชิฟ และอัลกอริทึม เราสามารถทำให้ใช้พลังงานได้อย่างเหมาะที่สุด อย่างไรก็ตาม เราไม่สามารถระบุเป็นที่แน่นอนได้ว่าระบบของเราใช้พลังงานต่ำกว่าระบบอื่นๆ

ในวิทยานิพนธ์ฉบับนี้ เราจะได้อธิบายในรายละเอียดเกี่ยวกับฮาร์ดแวร์และซอฟต์แวร์ที่ต้องใช้เพื่อการพัฒนาโครงข่ายที่เสนอนี้

ภาควิชา	วิศวกรรมไฟฟ้า	ลายมือชื่อนิสิต	[illegible]
สาขาวิชา	วิศวกรรมไฟฟ้า	ลายมือชื่ออาจารย์ที่ปรึกษา	[illegible]
ปีการศึกษา	2547		

4670621921: MAJOR ELECTRICAL ENGINEERING
KEY WORD: MOBILE NODE, TELE-MEASUREMENT, SENSOR NETWORK

LENG KY: DESIGN AND DEVELOPMENT OF CENTRAL NODE AND MOBILE NODE FOR TELE-MEASUREMENT SENSOR NETWORK. THESIS ADVISOR: ASSOC. PROF. WATIT BENJAPOLAKUL, D. ENG.

The conventional environmental network does not provide the feedback ability and is not interesting for the non-scientific user. So, we propose a personal sensor network for environmental monitoring and remote security system for company, organization and home user.

With its long-term data collection ability, the tele-measurement sensor network can be integrated in many research areas, especially in survey engineering, to collect the data with a very fine resolution comparing to the conventional method (human operator).

With its feedback ability, the network allows the end user for better monitoring and controlling their system more easily and reliably.

This network is easy to build and allows the end user to change the configuration and easy to update the system by using the modular technique and ability to set up new configuration from the center. It can be used as indoor or outdoor network.

By choosing appropriate microchip technologies and algorithm, we can optimize the power consumption of the system.

In this thesis, we describe in detail about hardware and software needed for developing this network (central and mobile nodes).

Department	Electrical Engineering	Student's signature	[illegible]
Field of study	Electrical Engineering	Advisor's signature	Watit
Academic year 2004			

Acknowledgements

This research is possible because of the help and support of numerous individuals. I wish to express my profound gratitude to all of them. I am greatly thankful to Associate Professor Dr. Watit Benjapolakul, the thesis advisor, for his continuous support, help and guidance throughout my time as his graduate student at Chulalongkorn University.

I deeply thank Dr. Somboon Chongchaikit (thesis chairman), Assistant Professor Dr. Chaodit Aswakul and Dr. Chaiyachet Saivichit (thesis committee) for their valuable comments and suggestions.

I shall always be grateful to my family (my parents, my brothers and sisters) for providing me with the best educational opportunities. They have been a source of incessant encouragement, which have been vital to all of my accomplishments.

This research was supported by the scholarship of AUN/SEED-Net and JICA. I wish to express my sincere thanks to the AUN/SEED-Net and JICA for their financial support. I also thank International School of Engineering, Graduate School of Chulalongkorn University and Department of Electrical Engineering of Chulalongkorn University for their great help.

I, particularly, thank Institute of Technology of Cambodia (ITC) for giving me the best opportunities to do this research through the AUN/SEED-Net and JICA project.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

LIST OF ABBREVIATION

ADC	Analog-to-Digital Converter
BCD	Binary Coded Decimal
CAN	Controller Area Network
CMOS	Complementary Metal Oxide Semiconductor
DAC	Digital to Analog Converter
DCE	Data Communication Equipment
DGPS	Differential Global Positioning System
DTE	Data Terminal Equipment
DTMF	Dual Tone Multi-Frequency
EEPROM	Electrically Erasable and Programmable Read Only Memory
GGA	Global Positioning System Fixed Data
GLL	Geographic position - Latitude / Longitude
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSA	GNSS DOP and Active Satellites
GSM	Global System for Mobile Communications
GSV	GNSS Satellites in View
I^2C	Inter-Integrated Circuit Bus
IC	Integrated Circuit
ICSP	In-Circuit Serial Programming
INT	Interrupt line
LCD	Intelligent Liquid Crystal Display
LED	Light Emitting Diode
M2M	Machine to Machine Communication
ME	Mobile Equipment Internal Storage
MN	Mobile Node
MSSP	Master Synchronous Serial Port
NRZ	Non Return to Zero
PBP	PICBasic Pro Compiler
PDU	Protocol Data Unit
POR	Power-on Reset
PSP	Parallel Slave Port
RAM	Random Access Memory
RISC	Reduce Instructions Construction Set
RMC	Recommended Minimum specific GNSS data

RMS	Root Mean Square
RTC	Real Time Control/Calendar
RTCM	Radio Technical Commission for Maritime Services
SA	Selective Availability
SCI	Serial Communication Interface
SCL	Serial Clock line
SDA	Serial Data Address
SIM	Subscriber Identity Module
SM	SIM Message Storage
SMS	Short Message Service
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SSP	Synchronous Serial Port
TTL	Transistor-Transistor Logic
USART	Universal Synchronous Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VTG	Course over Ground and Ground Speed
WAAS	Wide-Area Augmentation System

CHAPTER I

INTRODUCTION

1.1 Literature Review

Recent advances in wireless communications and electronics have enabled the development of low cost, low-power, multifunctional sensor nodes that are small in size and can communicate in short distances. The sensor network represents a significant improvement over traditional sensor and it can be used for various applications areas such as health, military, industry, home, etc. For different application areas, the sensor networks are designed with different technologies [1-2]. It is also a key technology to obtain user's context in the real world because it can enable long-term data collection at scales and resolution that are difficult, if possible, to obtain otherwise.

A large number of researchers have been working on sensor network technologies primarily for the realization of large scale environmental monitoring systems for military use and/or scientific use [1-4] and in this recent year, a large number of systems have been proposed for a small network and indoor network such as MICA [3] and U^3 node that is capable of communicating with other nodes to carry sensing data and queries issued by users.

Wireless sensor package developed by Ara N. Knaian can be used only for Intelligent Transportation Systems by counting passing vehicles, measuring the average roadway speed and detecting ice and water on the road [5]. However, they are not applicable to a non-scientific and home user and there is no feedback path.

A Linux-Based Robot Control System [6] developed by Traig Born and Joel Glidden is very powerful in terms of data collection and remote control, however, it cannot be considered as a sensor network because of its power consumption and its size.

Before we start our discussion about the proposed method, it is good for us to know what the requirements and characteristics of sensor network are and how they can be chosen, applied and combined with the existing technologies into the sensor nodes, which we can apply or integrate to the real word and how it can help improving our daily lives.

The requirements for sensor network are:

- Sensor nodes mainly use a broadcast communication paradigm.
- Sensor nodes are limited in power, computational capacities, and memory.
- Sensor nodes would have to be small, extremely robust, easily manageable and installable at a low cost [1].
- The ability to cooperate among the sensor nodes
- The topology of a sensor network changes very frequently.
- The sensor network should be able to support a high number of sensor nodes
- Easy to control and monitor all mobile nodes just from one place.

We will discuss first about the Design and Implementation of a Sensor Network Node for Ubiquitous Computing Environment (U^3: U Cube) and then A Wireless Sensor Network for Smart Roadbeds and Intelligent Transportation Systems and Computer Systems Senior Design.

1.2 Overview of "Design and Implementation of a Sensor Network Node for Ubiquitous Computing Environment"

Ubiquitous computing is a technology aims at supporting human activities with a number of computers and sensors that are deeply integrated into our daily lives. Such "smart" and "attentive" services would be realized using user's context and preferences in the real world [3].

U^3 is a 50 mm cube that contains a power module, a CPU module, an RF communication module and a sensor module. This U^3 is capable of sensing several types of data such as temperature, brightness, and the presence of human and sending this information to other nodes (via RF link) and/or peripheral devices including PC and PDA (via IR link).

1.2.1 Key Technology

Hardware Implementation

In order to achieve the sensor node requirement such as small size, low cost, low power consumption, multi-functionality and extensibility, the U^3 node is divided into four physically separated modules: power control module, processing module, communication module and sensing/actuating module as shown in Fig 1.

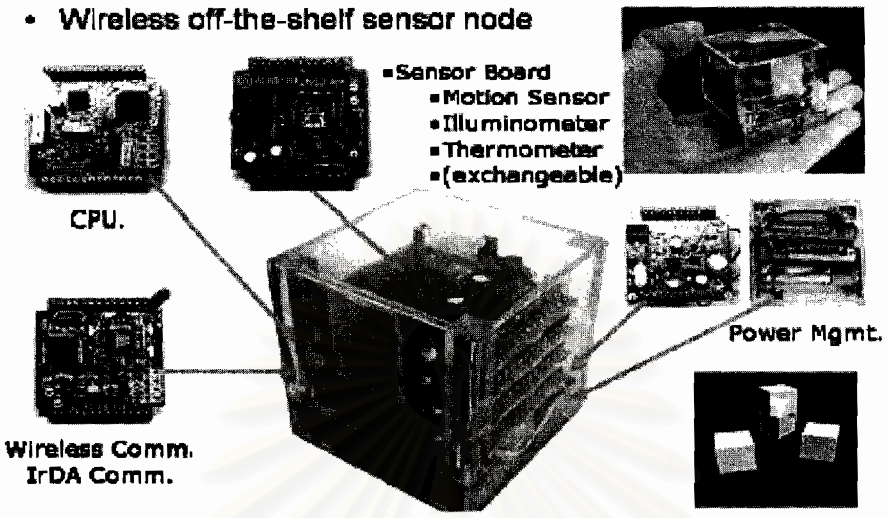


Fig 1. Hardware Components of U^3

Each module is connected to one another by a bus connector in order to achieve extensibility. By choosing an appropriate technology from Microchip [7], the U^3 can go to sleep mode when there are no events or tasks to operate, thereby achieves the low power consumption requirements. This requirement can also be achieved in software that we implement in the sensor node. That software must make efficient use of processor and memory. The sensor module consists of a motion sensor, a brightness sensor, and a thermometer. As the sensor module can be replaced, we can adopt the system to the new application easily, that is it fulfils multifunctional requirement. For the communication module, it consists of an RF monolithic transceiver CDC-TR-02B (300 MHz band, On-Off keying, 115.2 kbps), helical antenna, IrDA1.0 communication interface and one PIC Microcontroller for processing network protocol. IrDA1.0 communication interface is used as an interface between U^3 node and peripheral devices such as a PC or a PDA. Application programs and sensor data can be uploaded and/or downloaded through this connection. It is controlled by CPU module. All the necessary components are listed on TABLE I.

TABLE I
MAJOR COMPONENT FOR U^3

Sensor		Motion, Photo, Temperature
RF Transceiver		CDC-TR-02B (315MHz)
Micro Controller Unit		PIC 18F452
IrDA 1.0	Controller	MCP2150
	Transceiver	RPM851A
Calendar		RX-8564CF
Battery		Ni-MH

Software Implementation

In order to communicate among components, I^{2}C bus protocol has been implemented. In general, the data may be simultaneously captured from sensors, manipulated and streamed onto a network. Alternatively, data may be received from other nodes and forwarded in multi-hop routing or bridging situations.

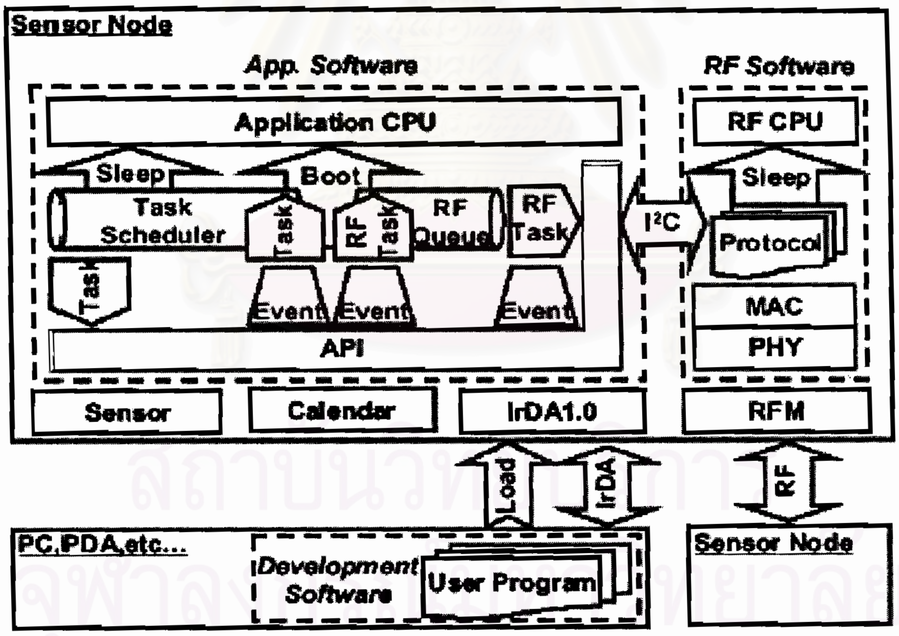


Fig 2. Software Architecture of U^3

To realize such concurrent execution within resource-restricted hardware, we allow interruption of tasks by events. Here, an event is defined as a process which has to be executed immediately and is assumed to complete immediately. Event includes the arrival of wireless packets, etc. On the other hand, a task is defined as a process, which takes longer time than an event to finish. It includes periodic capturing of environmental data, etc. Event-based task scheduling needs low overhead for state transition compared to stack based approach. Accordingly this scheme saves CPU load and power consumption. The task is controlled by the RTC (Real Time Control or Calendar Rx-8564CF). The software architecture is shown in Fig 2.

As we implement MAC (Media Access Control) algorithm and CSMA/CA protocol to achieve the collision avoidance and multiple accesses over RF wireless link, the accurate synchronization between sending and receiving nodes is crucial for high speed and reliable transfer. Accordingly, application tasks can be frequently interrupted by wireless communication events, so in order to remove such load from the CPU, we provide separate CPU for wireless communication and application tasks. For the packet format, see Fig 3.

1.2.2 Application and Performance Evaluation

The performances of the U^3 sensor node have been evaluated by developing a simple application that detects human motion in a room (Fig 4). As an experiment, the success ratio of packet transmission was measured. Six U^3 nodes were deployed in a 9m x 12m indoor environment. The sink node (connected to a data collection PC via IrDA) periodically queries the other nodes (source nodes) about motion sensor data over the RF wireless link. In response to the query, source nodes report to the sink node whether they detected a human motion.

```
typedef struct
{
uint16  addr;   // destination address
uint8   type;   // Application ID
uint8   group;  // Network ID
uint8   length; // Packet Length
uint8   data[DATA_LENGTH];
                //Payload (max:29bytes)
uint16 crc;     // CRC
} packet_type
```

Fig 3. Packet Format of U^3

Fig 5 shows each node's rate of successful transmission rate when the query repetition period is 0.5 sec, 1.0 sec, and 1.5 sec. The result shows that successful transmission rate is sufficiently high (91%) even when the query repetition period is 1.5 sec. This is a convergence of discarding sensor data packets on intermediate nodes if the nodes receive newer queries. Fairness of the successful transmission varies widely when the repetition period is large. This phenomenon results from the effect of packet loss over multi-hop networks.

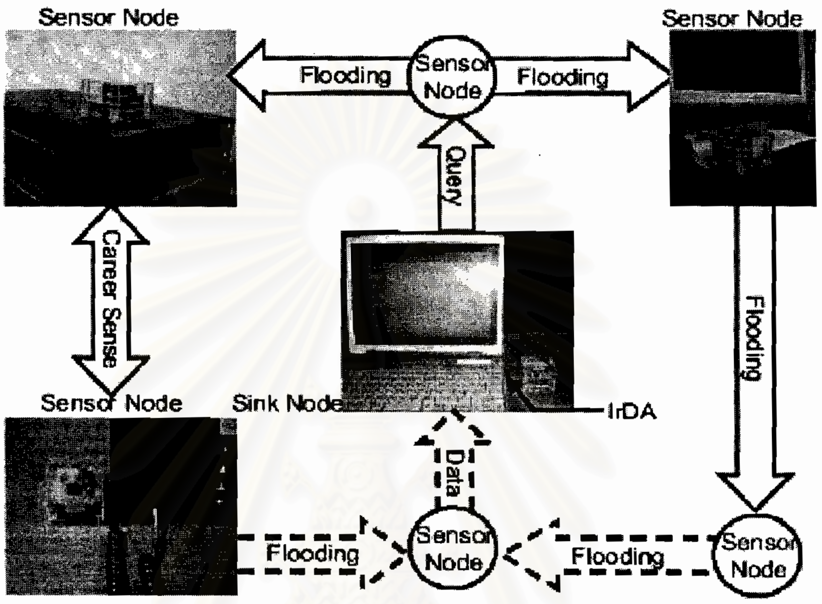


Fig 4. Experimental Application

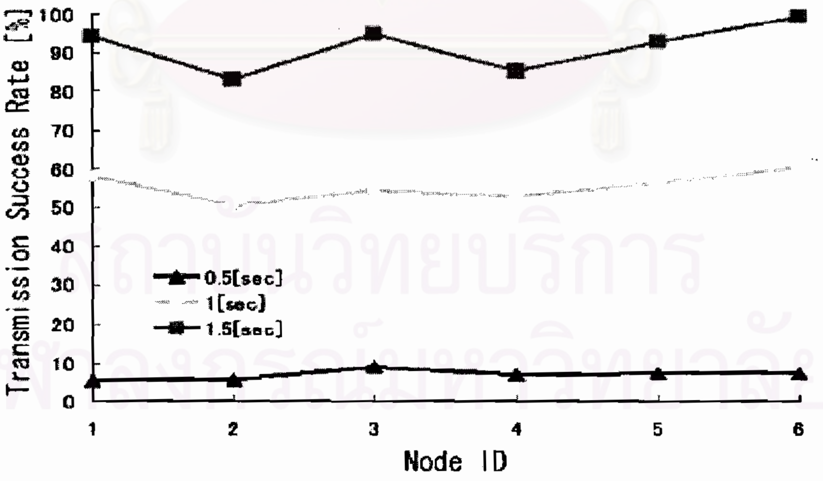


Fig 5. Transmission Success Rate

Fig 6 shows the success rate of data transfer when the number of nodes changes. In this experiment, each node tries to transmit a response packet that contains sensor data at random intervals over 100 seconds. Each packet is 13 bytes in length (including header and payload).

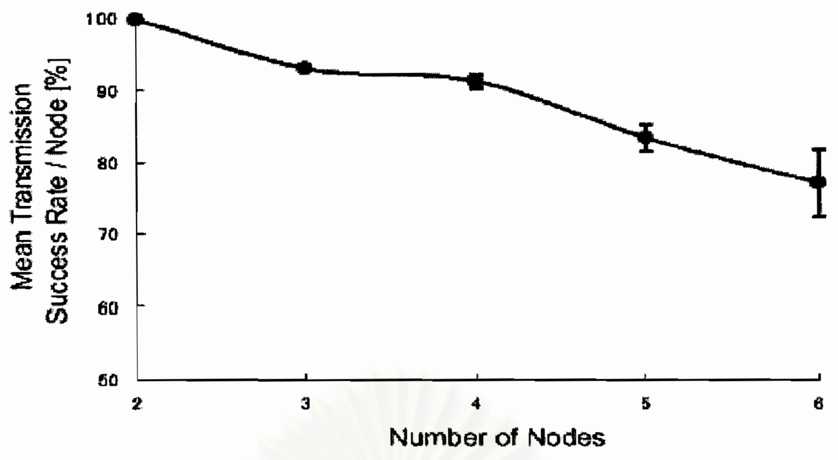


Fig 6. Mean Transmission Success Rate

Fig 6 shows that the success rate of the transmission varies widely as the number of nodes increases. This is due to the unbalanced random value used for the CSMA/CA back-off operation. As the operation to generate good random numbers is too expensive for low power microprocessor, only a simply random function for the CSMA/CA back off operation was used.

1.3 Overview of "Wireless Sensor Network for Smart Roadbeds and Intelligent Transportation Systems"

Because the congestion level on the nation's roadway is spiralling out of control, many techniques have been proposed to solve the problems. The solution was regrouped into two main methods, first group uses the information technology to make better use of the roads and the second is to make better public transportation, the latter is the only sustainable long-term solution to road congestion but requires a lot of investments. However, the information technology method will be discussed.

Typically, traffic sensor systems can be classified into two types. The first type uses sensors, placed at many points throughout the road network that count and measure the speed of passing vehicles. This data can be used to intelligently control traffic signals, and can be used with a simple queuing model to predict link times on the road network. The second type of system aims to measure link times directly, by tracking the progress of probing vehicles through the road network. In this article, the first type of system was used.

The most common type of traffic sensor is the in-road inductive loop. A coil of wire of several meters in diameter is buried under the road and connected to a roadside control box. The control box passes an electrical current through the coil. Passing vehicles change the inductance of the coil. In most units, the control box outputs a digital signal, based on a threshold inductance, to be used to control traffic signals or to count vehicles. Two loops, placed a few meters from each other, can be used as a speed trap. Traffic can also be detected using magnetic sensors that detect passing vehicles by measuring disturbances in the Earth's magnetic field. Magnetic traffic sensors are much more compact than inductive loop traffic sensors, and thus are better able to count vehicles in bumper-to-bumper traffic, so magnetic sensors was decided to be used in this system.

The design for a wireless in-road traffic sensor system was presented. The sensors are small, low in cost ($30), and extremely rugged. They count and measure the speed of passing vehicles using magnetic technology. They also measure information about road conditions. Each cluster of sensors transmits data to a receiver mounted on an electrical pole up to 300 meters away, which relays the data to a processing station. Each sensor node consumes so little power that it can operate from a small internal lithium battery for at least 10 years [5].

1.3.1 Key Technology

Each node is a compact, self-contained package. It contains magnetic field sensors, a temperature sensor, a radio transmitter, a microcontroller and a lithium battery (See Fig 7).

The node samples the magnetic field at its front and back ends (AMR: **A**nisotropic **M**agneto-**R**esistor), and internally processes this data to count vehicles and compute the average speed of passing vehicles. Once a minute, it transmits a data packet containing its ID, vehicle counts in 10-second bins, average speed, and temperature to the base station. All of these fundamental design choices were motivated by a desire to achieve the other design goals while making the most efficient use of power, to allow for as long a battery life as possible.

To detect vehicles, the system detects excursions of the sampled magnetic field value from a baseline. A passing vehicle generates an excursion first below and then above the baseline, (because it pulls field lines away from the sensor as it approaches and then pulls field lines onto the sensor as it drives over it) providing a simple, unambiguous way to detect multiple end-to-end vehicles. To measure the speed of a vehicle,

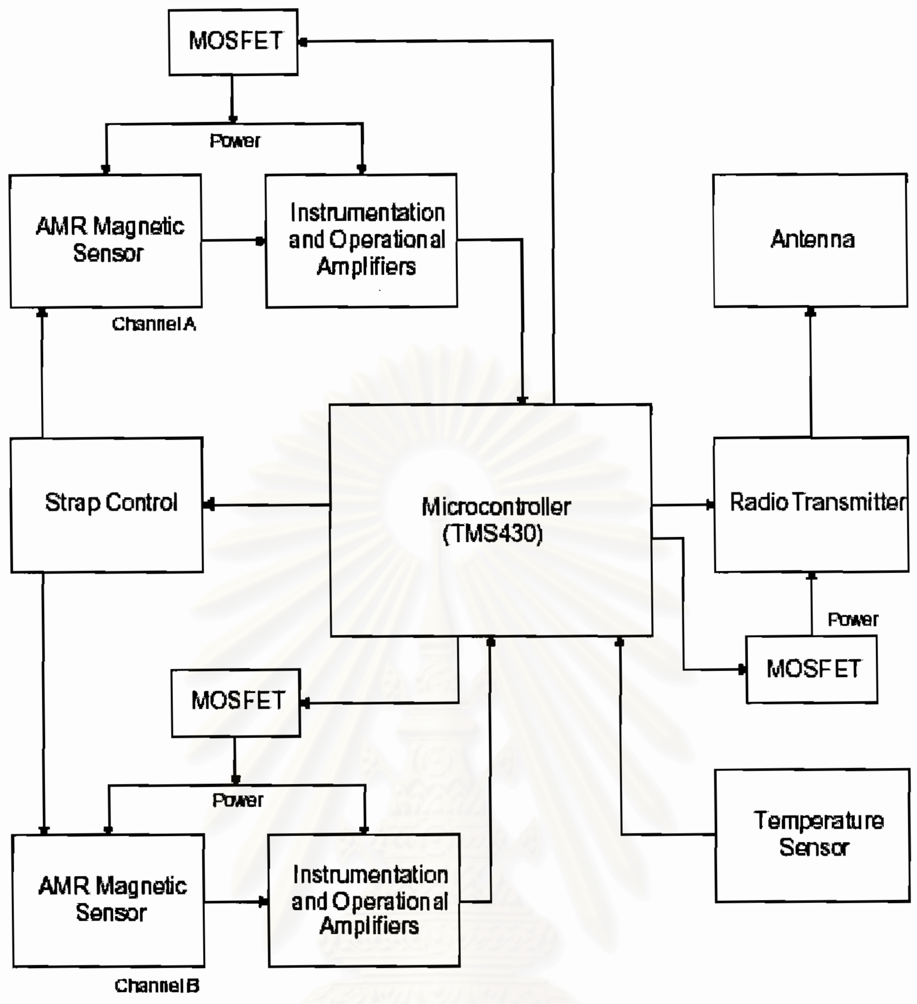


Fig 7. Node Block Diagram of Wireless Sensor Network for Smart Roadbeds and Intelligent Transportation Systems

the node waits until it detects an excursion from the baseline, and then starts sampling at 2 kHz using two sensors, one at the front of the node and one at the back. The waveforms at the outputs of the sensors are identical, except that they are shifted in time and may have slightly different electrical noise characteristics. The sensor waits for the signal from the rear sensor to cross the baseline, and then counts the number of samples until the signal from the forward sensor crosses the baseline; it is like what we use in frequency meter. From this count, because we know what the distance between two sensors is and how long it takes to pass this section, it can readily compute the speed of the passing vehicle.

Since the sensor package is to be installed directly into the roadbed, it can also measure information about road conditions, such as whether the road is covered with snow, ice, or water. In order to do this, we use a capacitive sensor by attaching a micro-controller output pin to an electrode near the surface of the sensor package, and attaching an adjacent electrode to a micro-controller A/D input through a simple FET amplifier. The microcontroller would repeatedly apply a voltage step to the electrode, wait for a varying number of microseconds, and then sample the A/D input. By sliding the closing time of the A/D sampling gate across the ring-down from the step, the microcontroller can use synchronous demodulation to make the capacitive measurement.

Another approach is to measure the temperature: If the temperature is above freezing, ice cannot be present; similarly, if the temperature is at or below freezing, ice may be present. In addition, the rate of change of the in-road temperature versus the air temperature is determined by the heat capacity of the material above the sensor.

Each node has a transmitter, but no receiver. This design choice was made primarily to cut the cost and complexity of each node, since the nodes do not have any intrinsic need to receive commands. However, the lack of a receiver complicates the radio protocol, since most standard radio multiplexing techniques require each participant to be able to receive as well as send. The protocol designed is very simple: each node randomly selects a time slot within a 60- second interval and transmits there. If it detects that a vehicle is present when its randomly selected transmission time arrives, it waits for it to pass before transmitting. Redundancy and sparse use of the channel reduce the probability of collisions to an acceptable level. To increase the number of nodes per base station, several frequencies can be used, with a frequency deterministically assigned to each node. This combines both TDMA and FDMA. In practice, in order to reduce the chance of collision, we assign only 10 time slots in one frequency so in total, we can assign up to 80 nodes per base station (each transmitter module can support 8 channels from 902 MHz to 928MHz ISM band at a maximum rate of 50 KB/s).

The radio transmitter used on the node requires an edge on the digital input data at least every 33 ms; otherwise PLL may track out the modulation. So, each transmission requires a one-character preamble to take the PLL from lock to capture. This character may not be transmitted correctly, so it is not protected by the packet's CRC. Fig 8 is a diagram of the packet sent by the node to the base station. The first character of the packet is a magic number to identify the system type and software version. The next six bytes are the node ID. Each node has its own unique ID to simplify administration of the network. The last 18 10-second count bins follow, (so that even if one message is lost, the counts are not lost), followed by the average speed. The message concludes with a CCITT CRC-16. The total message length (including the character to put the PLL into capture mode) is 30 bytes, which takes 7.8 ms to transmit at 38,400 kb/s.

PLL Capture Character 'U'	Magic Number 'P'	Node ID (6 Bytes)	Vehicle Counts 18 10-Second Bins (18 bytes, least to most recent)	Average Speed (1 Byte)	CCITT CRC-16 (2 Bytes)

Fig 8. Packet Format

1.3.2 Test Results

According to the visual observation, it confirmed that the sensor worked as expected. Generally, it counted vehicles that drove over it, and did not count vehicles that swerved around the pothole (a small fraction) or that were traveling in other lanes. The sensor double-counted certain types of large trucks, and occasionally (perhaps once every twenty) did not count a vehicle. The sensor performed just as well in bumper-to-bumper traffic as it did in widely separated traffic.

1.4 Overview of "Computer Systems Senior Design"

This topic seems to be different from the above two topics, however, it is very important to understand it because it also integrates some sensor network characteristics and relates to our research topic as well.

First of all, some basic mechanism of what so called Robot would be introduced. Robots typically contain three interacting sub systems such as: *Mechanical subsystem*, *Control & Sensory subsystem* and *Behavioral subsystem*. Since many researchers work in the mechanical or behavioral areas and since the control system ties the mechanical and behavioral aspects together to be a complete robot, only the control system will be discussed.

The objective of this robot control system is to be general purpose, flexible enough for a wide range of applications and inexpensive enough to be accessible for small educational institutions [6].

Typically, a robot control system musts satisfy four functional requirements [6]:

1. Provide a communication mechanism for remote control, monitoring and configuration.
2. Acquire data from sensors.
3. Perform computations on acquired data according to programmed behaviors or artificial intelligence.
4. Generate output signals based on these computations to drive the robot's mechanical subsystem.

For these requirements, it resembles our work very much that's why we cannot skip it.

Key Technology

Since the Linux is an open-source operating system and available at no cost via the Internet; it is decided to be used as an operating system for the control system board (called Rox). For the program and acquired data, they are stored it in Flash drive because it consumes low power, contains no moving part that lead it to be the best choice for mobile robotic application where vibration and limited power are concerned.

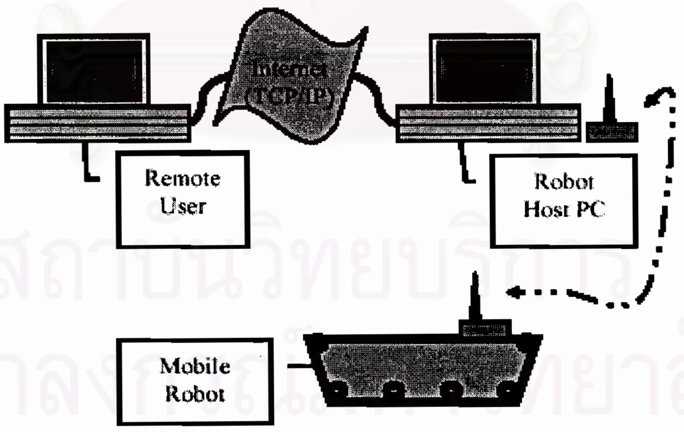


Fig. 9. Rox's Communications and Control

Communications: As a general-purpose robot controller, Rox must provide a communication mechanism suited to many different applications. Perhaps Rox is to be used on an autonomous robot that collaborates with other robots to accomplish some tasks. Perhaps Rox is to be used on an Unmanned Air Vehicle conducting battlefield surveillance while being piloted by remote control. See Fig 9. Rox is located on the mobile robot and linked to a host computer via 900 MHz radio modems. This host computer then serves as an operator station and connects Rox to a larger network such as the Internet. Rox's host also handles network security ensuring that only authorized users gain access.

Data Acquisition: Because Rox is based on standard PC hardware, there are a number of options available for data acquisition. Sensors can be easily interfaced to COTS (Commercial Off-The-Shelf) data acquisition boards that connect to USB, ISA or PCI interfaces. The parallel port provides another sensor interfacing option. Some sensor data can be acquired through Mox (**M**otor control system used by R**ox**), which monitors sensors connected to the robot mechanisms to facilitate motion control. See Fig 10.

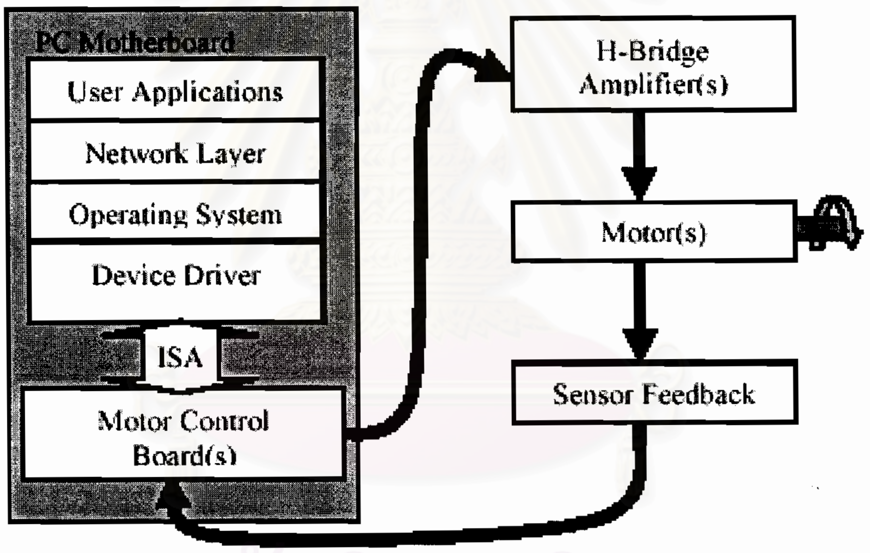


Fig 10. Mox High-Level Block Diagram

Computation: As Rox is essentially an embedded computer system, it is composed of the major components of most personal computers such as: Motherboard, Microprocessor, Memory, Long-term storage and Operating System. A standard PC motherboard with SDRAM and a Pentium 133 microprocessor provides an inexpensive computing platform suited to almost any robotic application.

Output to Mechanical Subsystem: Rox provides drive signals to the robot's mechanical subsystem via its custom designed motor control system (Mox). Mox integrates hardware and software components to translate computation into actuation. The system has four major functional components:

1. Motor Control Board } Motor Control Board(s) in Fig. 10
2. Motor Control Board Device Driver }
3. Motor Control Programming Library
4. **H-Bridge** Amplifier (**4** Darlington **configured** NPN power transistors)

The relationship among these components and to Rox is shown in Fig 10.

This thesis is organized as follows. In Chapter 2 we describe our proposed solution that bases on real experiment and implementation. Chapter 3 describes the implementation of our proposed system in more detail including component study, format, flowchart for each algorithm and some tricks that we use while implementing this system. The performance evaluations of each features and comparison between the original data in the memory and the received data from mobile phone are interpreted in Chapter 4. Finally Chapter 5 concludes our work and the recommendation.

1.5 Scopes and Goals

Scopes:

- Implement the mobile node by using PIC as controller and central node by using PC as controller.
- Evaluate the system performance by comparing the original data in the memory and the received data from mobile phone for exchange algorithm testing.
- Evaluate the system performance by adding some error in the received data or interrupt the mobile node while central node want to connect to the mobile node to test the central node performance and scheduler algorithm.
- Evaluate the system performance by making the input reach the critical condition to test the mobile node performance and script algorithm.

Goals:

- Contribute in rural development and survey engineering with our long-term data collection capability and fine resolution capability.
- Contribute in remote security system control by scanning all the input ports and the ability to report the data when it reaches the critical value.
- Suitable for indoor and outdoor network because we choose mobile phone system as our communication link.
- Capability to change the configuration setting of the system from remote distance.
- Reduce the expense of the budget because we only spend the money for equipment, no need to pay for the staffs every month.
- Can be in hand for all kinds of users (company, organization, military, scientist, student, home user, etc.) because they can easily manage the network during setup as well as during operation process.

1.6 Expected Benefit

- Give an understanding on how to send data through mobile phone, how to use AT-Command and how the message data is packaged into 8 bits ASCII before sending (for SMS mode) and why we need to convert it, etc.
- Give an understanding on how to arrange the memory efficiently in order to be easy when we want to retrieve data from memory, etc, how we can update the configuration of the system by just updating some data in that memory.
- Give an experience on how to choose/select the format for storing/sending data when the data cannot be represented by the whole ASCII character set.
- Give an experience on how to choose/select the Operational Amplifier (OPAmp) for the interface board (some sensors need to use amplifier in order to be able to connect to the microcontroller) because some OPAmps cannot be used with the power supply 5V, low power, high input impedance and rail to rail input/output voltage, etc.

CHAPTER II

PROPOSED SYSTEM

As descript above, other systems can not be in hand for non scientific user and can not update easily whereas our proposed system is a private network, can be in hand for scientific as well as non scientific user, easy to update and allows the user to change the configuration by itself. It is an outdoor and indoor network that can be used for a large or small-scale network such as environmental monitoring and remote security system. It is very difficult to envision how such future application should be like, however it is desirable to specify or list some applications, so we try to show some real applications as listed below:

- Measure the water level, gas toxic, air pollution rate, humidity, and luminosity and monitor these values from distance.
- Measure the current intensity of the power distribution sub center, alert the main center with the cause and cut off the dangerous line at the exact place.
- Use as security guard: monitor the gate, doors, windows, hallways, alert the security office and/or police stations with the location where the unpleasant event have been occurred.
- Control the electrical appliances in the house.

2.1 Hardware & Software Design for Mobile Node

2.1.1 Hardware

In order to fulfill the flexible requirements, we divide the system into small modules (modular system), with have specific functionalities, as shown in Fig 11.

Radio Link Module: Use a mobile phone kit as the radio link module to ensure the data transfer between central and mobile node (MN). We use one PIC16F876 [7] (Programmable Integrated Circuits) and one EEPROM 24C512 [8] (Electrically Erasable Programmable Read Only Memory) to ensure this functionality.

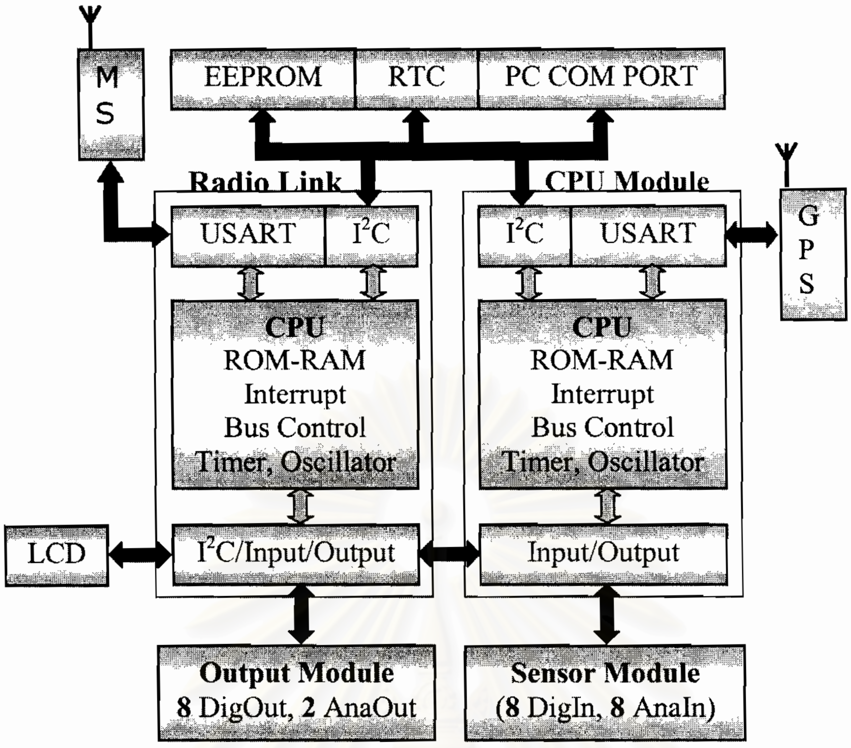


Fig 11. Mobile Node Architecture.
MS: Mobile Station, GPS: Global Positioning System Receiver.

CPU Module: use PIC16F877 [7] to ensure the data processing, sampling and storing data in the external EEPROM. This module communicates with the Radio link module, some external sensor boards and Real Time Control (RTC) DS1307 [9] via I^2C protocol.

Sensor Module: As the CPU module supports digital & analog input, some sensors can connect directly to digital input port and other sensors that give the result as voltage (0V to 5V such as humidity sensor, luminosity sensor (use photo-resistor), Pressure sensor, etc.) can connect directly to analog input port. We also reserve one RS232 port for GPS receiver for positioning functionality.

Output Module: We fix to use I^2C-Digital to analog Converter PCF8591 [10] to provide two analog outputs for the user with 8 bits resolution and PCF8574 [10] to provide 8 digital outputs. The buffer amplifiers for analog outputs (to protect our DAC) and power switching for digital outputs (for 12V and/or 220V load) is implemented but will be add to this mobile node as necessary.

Power Module: will be developed by using solar panel if necessary. Right now, we use adaptor AC~9V to supply the entire system.

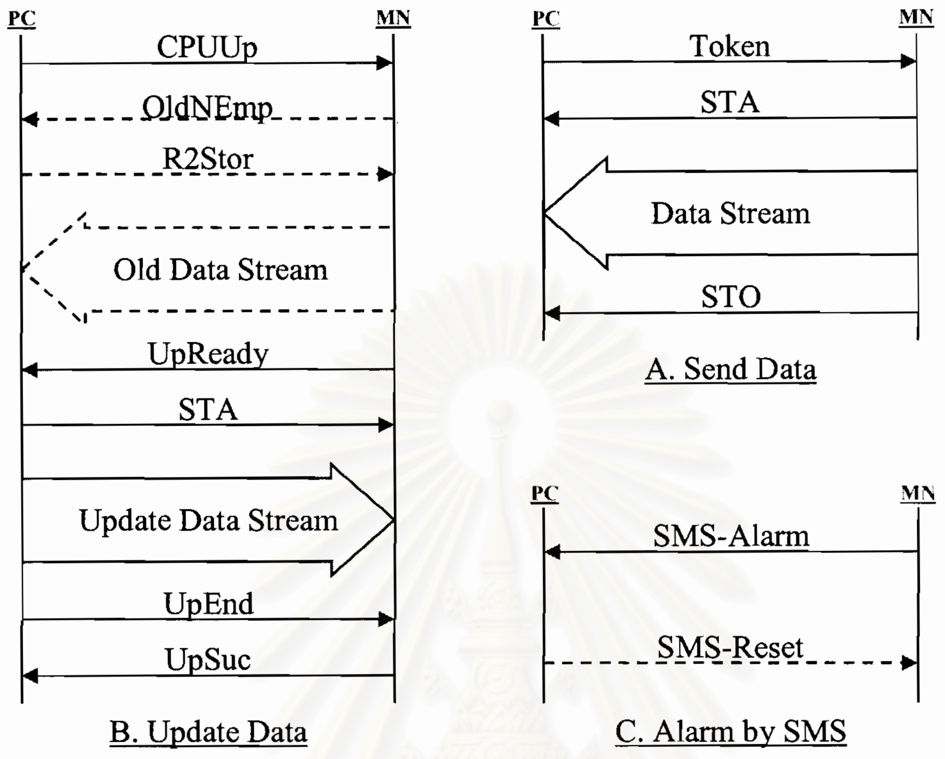


Fig 12. Time diagram for the Exchange of Command and Response Signaling

Note: All these features are supported by our CPU module. If we want to use it, just connect the corresponding module then enable it from PC (Personal Computer). This implies flexibility and saving money because we pay for what we want. The modular technique provides not only flexibility but also power consumption reduction as the CPU module can go to sleep mode (I<2 mA at 5V, 4 MHz) after processing all its works (it take about 300 ms/scan, so if we scan the input every 1 min, the CPU can go to sleep mode for 59.7s that is equivalent to 99.5% of time).

2.1.2 Software

We implement a layer-2 protocol to Mobile node in order to archive the flexibility to change the configuration by the user during or after the network configuration. We also provide the feedback control by implementing a script execution function. The program is written by using **PicBasic Pro Compiler** [11-14].

TABLE II
COMMAND AND RESPONSE SIGNALLINGS

Name	ASCII Character	Description
STA	178	Indicate the Start of data
STO	187	Indicate the End of data
Token	196	Request for Sending data
CPUUp	205	Request for Updating data
R2Stor	214	Ready to store the old data
UpEnd	223	Indicate the End of Update-Data
OldNEmp	232	Data left in EEPROM
UpReady	241	Ready to store Update-Data
UpSuc	250	Updating is finished successfully
S	167	Synchronizing Frame Header

Radio link module: We use AT-Command to send the data and SMS (Short Message Service) via Air-interface (mobile phone), however, as not every mobile phone supports text mode; we implement text-to-PDU (Protocol Data Unite) mode algorithm [15] for we can support the majority of mobile phone. We also implement a layer-2 protocol on it in order to ensure data transfer between the central and mobile node as shown in Fig 12. Send the collected data, update the user customization/setting, alarm warning ("Critical input detected!" and "The memory is full at mobile node: RobotXXX"), send input data, feedback control and send its report are the functionality supported by this module. For more information about the format of these messages, see section 3.2.6. All information about this protocol is shown in TABLE II with other constants used in our work.

Note: All the user customization, warning message and acquired data are stored in the external EEPROM, which is AT24C512 (512kbits ⇔ 64kbytes). This EEPROM (0000h→008Fh) stores, not only the data for *Radio link module* (mobile node ID, central mobile phone number, manager mobile phone number, SMS option, digital output port's names, analog output port's names), but also a very useful data for *CPU module* (options for CPU: scanning period, frame size, GPS flag, input mask, input critical values, upper bound counter limit and alarm flag, analog comparison status).

Here, the sampling period is the same for all input ports because we do not know exactly the future application of each port. After the network configuration, the user can only change the options for CPU. By just changing the data at the corresponding memory location, we can allow the user to customize the network very easily during installation as well as after installation. The memory location from 0090h to 00FFh is used for alarm and report message, from 0100h to 011Fh is used to store script's keyword, from 0120h to 0195h is used as report format for "READ" command, from 01A0h to 013Ch is used as report format for "REPORT" command and from 01C0h to FFFFh is used for storing acquired data. Section 3.2.6 will discuss more detail about memory location and script command keyword.

CPU Module: Ensure the data processing function. We sample the input port every sampling period (integer value from 1 to 60 second or minute) and store the data in the external EEPROM. Note that not all the inputs are sampled; we sample only the valid input ports defined by the input mask (*reduce power consumption and storage*). We suppose that the digital input and all the analog inputs are enabled. In case we have, let say, 2 analog inputs, our data per each sampling contains 2 bytes of digital input, 2 bytes of analog port #0 and 2 bytes of analog port #1.

The fact that we set bit #6 of each byte to 1 is to avoid the data from special character such as character 0, character 10, character 13, and character 26, etc. these characters are considered as a command, not a data, for modem and cause modem to disconnect without sending disconnect command because the data themself have the same meaning as a command. For GPS's data, we store it directly without any modification because it is printable ASCII code already.

Sending data process is controlled by the central node, it looks like "one to many" relationship, only the center computer can ask and the other nodes can only answer. In case that the CPU module stores the data more than CCCCh, the CPU module will inform the Radio link module to send the SMS with the message "*The memory is full at mobile node:* **ID**" where **ID** is the mobile node itself. At the same time, the CPU continue to store up the data in that memory until it reach the address FFFE0h, at this time, the CPU will restart to store the new data from address 01C0h, however, this is very rare because we already thought about this problem and we finally found the algorithm then program it at the center computer. See also Fig 13.

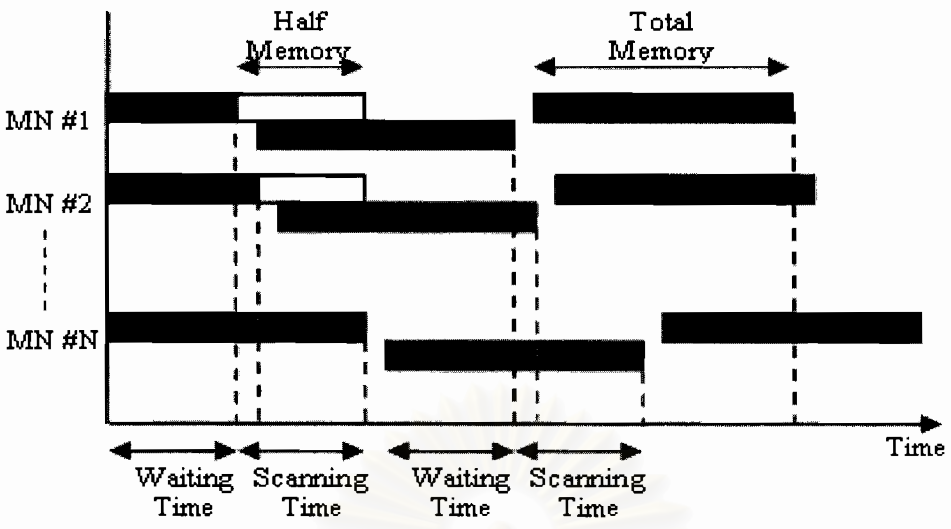


Fig 13. Scanning Process Chronogram

In case that any input data reaches the critical value (data is smaller/bigger than threshold value, predefined by the user, for analog signal or data is different from user predefined value for digital signal), we count up the counter by 1. We continue to count up whenever any input is critical and the counter value is less than the upper bound counter limit. When all inputs change to a normal value, we count down the counter by 1 and continue to count down if all the next sampling inputs are normal and the counter value is bigger than 0. Assume now that the counter reaches the upper bound counter limit, the CPU Module will tell the Radio link module to send alarm message via SMS with the current input value from all ports beginning by the message "*Critical input detected!*" and then reset the counter. After the center computer received the alarm message, it can send the feedback command via SMS (script format) to the mobile node to activate some output ports and reset alarm (it's up to the user customizations). Script format is defined by character 60 and character 62 in ASCII mode. Any text begins with ASCII character 60 and ends with character 62 will be considered as command word, for example: <RESET> is used to reset alarm. The reason why we implement this format is to avoid the accidental error when unknown user sends a wrong message to this mobile phone. In case that we do not send the reset alarm, the mobile node will continue to report the critical port whenever the counter reaches the Upper bound.

2.2 Hardware & Software design for the Center Computer (Central Node)

2.2.1 Hardware

We need one mobile phone connected to the COM port. Right now, we use ERICSSON T68. We implement a Serial-to-I^2C interface adapter for the Central node in order to provide the end-user the ability to customize their network during installation. Section 3.1.12 will describe how this board was designed.

2.2.2 Software

We implement the same layer-2 protocol (as for Radio link Module at Mobile node) and I^2C (Inter-Integrated Circuit Bus) protocol in the central node by using the VB programming language (Visual Basic). Moreover, we create GUI (Graphic User Interface) [16] and database (user Application layer) for storing the mobile node information such as mobile node ID, phone number, location (province, district, section, and description), input and output port's name and its critical values, etc. A database is a place where we store information and we organize the information in our database in such a way that our programs and applications can provide useful functions for end-users [17]. For GUI, we also provide the ability to the user to customize their own network by changing the Input mask, Telephone number, etc, in short, the user can change the data needed for Radio link module and CPU module (0000h→008Fh) during the network installation. We also provide some functionalities to the user to save the input data automatically to the user-predefined-folder, plot the record data like in MATLAB or Excel, print the record, show how many mobile nodes this system can support with the current configuration, etc. The latter is implemented to avoid the problem of not enough memory to store new data at mobile node. After reading the message, the program will automatically delete the message from the center computer's mobile phone preventing out of memory in the center computer's mobile phone.

Note: If we set Sampling Period to 0, it means that we do not want to save the input data but scan as fast as possible (every 0.2s). This mode is useful for the example application #4, #5 and #6. In fact, we can sample the input faster than this; however, as we want to save the power and 0.2s sampling period is fast enough, it was decided as a sampling period for these applications.

To determine how many mobile nodes this system can support and how long we can store the data before send it to PC, the following formulas are used.

$$N_F = M / ((F_S*2) + H) \quad (1)$$

$$T = N_F*T_S*F_S \quad (2)$$

$$N = T/T_0 \quad (3)$$

Where:

N_F :	Number of Frames contained in the last half memory.
M :	Half Memory size constant. It is about 32544 bytes for 64 Kbytes memory.
F_S :	Frame size defined by the user.
H :	Time Header. It is used to indicate the beginning of new frame. H = 7 bytes.
T :	Time needed to fill all that half memory.
T_S :	Scanning period defined by the user.
T_0 :	Scanning time needed for 1 node. It is equal to 4 min for 64 Kbytes memory.
N :	Number of Mobile Nodes supported by this system.

The longest Scanning time must be less than or equal to the Waiting time. See Fig 13 for more detail.

Now let us assume that the Frame-Size is set to be 60, Time-Scan to 1 min, EEPROM size is 64 Kbytes, only one analog input port is activated. By a simple calculation, we found that our system can finish scanning all mobile nodes within 15480 min and can support up to 3,870 mobile nodes. However if our system, with the same assumption, just have 15 mobile nodes, we need only 15*4 min. = 60 min. for scanning all the mobile nodes but the period of scanning is 15,480+60 = 15,540 min; it means that we scan only 60 min and after that wait for 15,480 min. for the new scanning process.

For the following chapter, everything in this chapter will be described in more detail especially, hardware implementation.

CHAPTER III

SYSTEM IMPLEMENTATION

Now, let us explain about our system in detail. This chapter covers all the necessary information for hardware, software and the way to use it. Some experiment scenarios are included in the next chapter but its interface board is included here.

3.1 Component Study & Hardware Design

3.1.1 PIC 16F87X and Modular Technique

PIC 16F87X is a product from MICROCHIP Company, which has many features as descript in the datasheet [7]. Among these features, we focus only on:

- High-performance RISC (Reduce Instructions Construction Set)
- In-Circuit Serial Programming (ICSP) via two pins
- Low power consumption (Power consumption < 2 mA typical at 5V, 4 MHz and Power consumption < 1 μA typical standby current)
- Power-on Reset (POR)
- Programmable code-protection
- Power saving SLEEP mode
- Low-power, high-speed CMOS FLASH/EEPROM technology
- 10-bit multi-channel Analog-to-Digital converter
- Synchronous Serial Port (SSP) with Serial Peripheral Interface (SPI) in Master Mode and Inter-Integrated Circuit Bus (I^2C) in Master/Slave mode
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection

For this project, PIC16F876 is chosen to be a core component for Radio Link Module and PIC16F877 is chosen to be a core component for CPU Module. Both of them have 8K of FLASH Program Memory (14-bit words), 368 Bytes of Data Memory (RAM) and 256 Bytes of EEPROM Data Memory and operate at speed up to 20 MHz clock. However, 4 MHz clock is chosen because the speed is fast enough for our application and it can reduce the power consumption as well comparing to 20 MHz (reduce the number of instruction per second is equivalent to reduce the power consumption).

The reason why PIC16F876 is chosen to be a Microcontroller for Radio Link Module is that it has big Program Memory to store the program code, big RAM to store incoming PDU-SMS (Short Message Service in Protocol Data Unit format) & other program variable, big EEPROM to store incoming Text-SMS (SMS in text format) & outgoing PDU-SMS and its input/output pin is enough to communicate with Mobile phone, LCD (intelligent Liquid Crystal Display), I^2C peripheral (external EEPROM, RTC and Output Module). With similar reason, PIC16F877 is chosen to be a Microcontroller for CPU Module.

In fact, PIC16F87X is not the most recent product from Microchip. We can find the newer product such as PIC17FXXX and PIC18FXXX that can handle all PIC16FXXX features and also have more features such as Program Memory, RAM, EEPROM and number of Interrupt Sources is bigger; USB, RS485 and Controller Area Network (CAN) are supported. The reason is that we develop our program on PICBasic Pro Compiler (PBP) version 2.30A [13], which is limited in term of keyword and PICmicros series 17FXXX and PIC18FXXX. Another reason, PIC18FXXX just well know in Thailand after we nearly finish our project but the main reason is that we want to use modular technique in this project. As we descript in chapter 2, modular technique can provide us the flexibility to adapt to the new project and save money, because we only use and pay for what we want, and also provide the flexibility to update the system because we can change one module without interfere other module.

The following paragraph will explain about the peripheral, which is used in this project. It include USART Asynchronous mode, Analog-to-Digital Converter (ADC), I^2C Bus and I/O Port.

USART Asynchronous Mode: In this project, we use standard non-return-to-zero (NRZ) format (one start bit, eight data bits and one stop bit) for both RS232 port (Mobile phone and GPS module). With 4 MHz clock, we can achieve 0.16% error at 9600 bps as shown in the TABLE III.

TABLE III
BAUD RATE FOR ASYNCHRONOUS MODE

BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 0)

BAUD RATE (K)	FOSC = 20 MHz			FOSC = 16 MHz			FOSC = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	1.221	1.75	255	1.202	0.17	207	1.202	0.17	129
2.4	2.404	0.17	129	2.404	0.17	103	2.404	0.17	64
9.6	9.766	1.73	31	9.615	0.16	25	9.766	1.73	15
19.2	19.531	1.72	15	19.231	0.16	12	19.531	1.72	7
28.8	31.250	8.51	9	27.778	3.55	8	31.250	8.51	4
33.6	34.722	3.34	8	35.714	6.29	6	31.250	6.99	4
57.6	62.500	8.51	4	62.500	8.51	3	52.083	9.58	2
HIGH	1.221	-	255	0.977	-	255	0.610	-	255
LOW	312.500	-	0	250.000	-	0	156.250	-	0

BAUD RATE (K)	FOSC = 4 MHz			FOSC = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	0.300	0	207	0.3	0	191
1.2	1.202	0.17	51	1.2	0	47
2.4	2.404	0.17	25	2.4	0	23
9.6	8.929	6.99	6	9.6	0	5
19.2	20.833	8.51	2	19.2	0	2
28.8	31.250	8.51	1	28.8	0	1
33.6	-	-	-	-	-	-
57.6	62.500	8.51	0	57.6	0	0
HIGH	0.244	-	255	0.225	-	255
LOW	62.500	-	0	57.6	-	0

BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 1)

BAUD RATE (K)	FOSC = 20 MHz			FOSC = 16 MHz			FOSC = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	-	-	-	-	-	-	-	-	-
2.4	-	-	-	-	-	-	2.441	1.71	255
9.6	9.615	0.16	129	9.615	0.16	103	9.615	0.16	64
19.2	19.231	0.16	64	19.231	0.16	51	19.531	1.72	31
28.8	29.070	0.94	42	29.412	2.13	33	28.409	1.36	21
33.6	33.784	0.55	36	33.333	0.79	29	32.895	2.10	18
57.6	59.524	3.34	20	58.824	2.13	16	56.818	1.36	10
HIGH	4.883	-	255	3.906	-	255	2.441	-	255
LOW	1250.000	-	0	1000.000		0	625.000	-	0

BAUD RATE (K)	FOSC = 4 MHz			FOSC = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-
1.2	1.202	0.17	207	1.2	0	191
2.4	2.404	0.17	103	2.4	0	95
9.6	9.615	0.16	25	9.6	0	23
19.2	19.231	0.16	12	19.2	0	11
28.8	27.798	3.55	8	28.8	0	7
33.6	35.714	6.29	6	32.9	2.04	6
57.6	62.500	8.51	3	57.6	0	3
HIGH	0.977	-	255	0.9	-	255
LOW	250.000	-	0	230.4	-	0

To set up an Asynchronous Transmission, the following steps are needed:

1. Initialize the SPBRG register for the appropriate baud rate. If a high-speed baud rate is desired, set bit BRGH (TABLE IV).
2. Enable the asynchronous serial port by clearing bit SYNC and setting bit SPEN.
3. If interrupts are desired, then set enable bit TXIE.
4. If 9-bit transmission is desired, then set transmit bit TX9.
5. Enable the transmission by setting bit TXEN, which will also set bit TXIF.
6. If 9-bit transmission is selected, the ninth bit should be loaded in bit TX9D.
7. Load data to the TXREG register (starts transmission).
8. If using interrupts, ensure that GIE and PEIE (bits 7 and 6) of the INTCON register are set.

Fig 14-A and Fig 14-B show the USART Transmit block diagram and Asynchronous Master Transmission respectively. TABLE IV shows the Register Associated with Asynchronous Transmission.

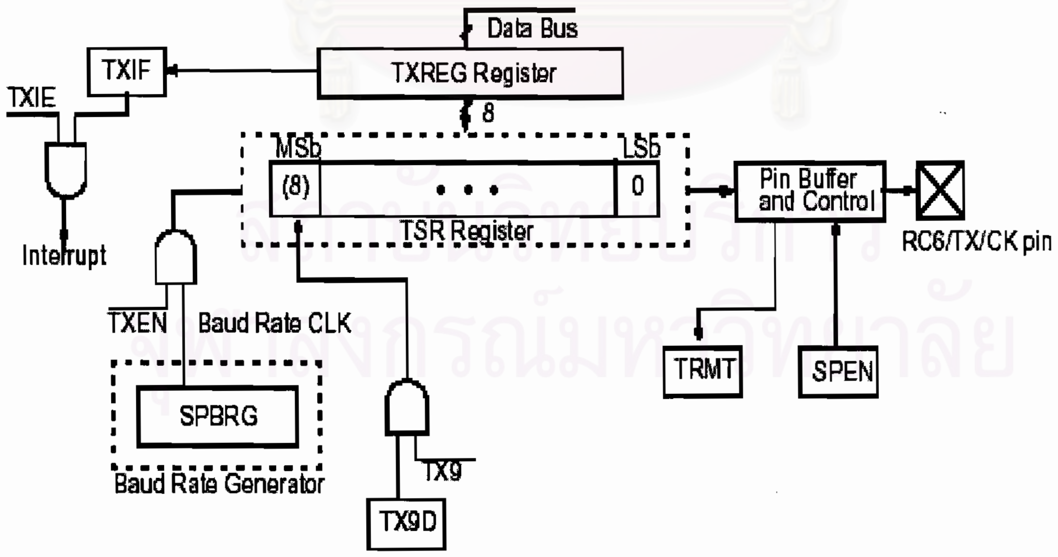


Fig 14-A. USART Transmit block diagram

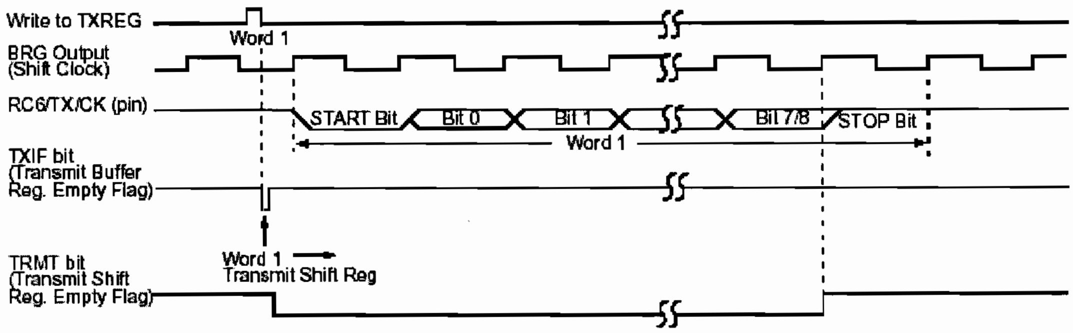

Write to TXREG
Word 1
BRG Output (Shift Clock)
RC6/TX/CK (pin)
START Bit
Bit 0
Bit 1
Bit 7/8
STOP Bit
Word 1
TXIF bit (Transmit Buffer Reg. Empty Flag)
TRMT bit (Transmit Shift Reg. Empty Flag)
Word 1 → Transmit Shift Reg

Fig 14-B. Asynchronous Master Transmission

TABLE IV
REGISTER ASSOCIATED WITH ASYNCHRONOUS TRANSMISSION

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
19h	TXREG	USART Transmit Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented locations read as '0'. Shaded cells are not used for asynchronous transmission.
Note 1: Bits PSPIE and PSPIF are reserved on the PIC16F873/876; always maintain these bits clear.

To setup an Asynchronous Reception, the following steps are needed:

1. Initialize the SPBRG register for the appropriate baud rate. If a high-speed baud rate is desired, set bit BRGH (TABLE V).

2. Enable the asynchronous serial port by clearing bit SYNC and setting bit SPEN.

3. If interrupts are desired, then set enable bit RCIE.

4. If 9-bit reception is desired, then set bit RX9.

5. Enable the reception by setting bit CREN.

6. Flag bit RCIF will be set when reception is complete and an interrupt will be generated if enable bit RCIE is set.

7. Read the RCSTA register to get the ninth bit (if enabled) and determine if any error occurred during reception.

8. Read the 8-bit received data by reading the RCREG register.

9. If any error occurred, clear the error by clearing enable bit CREN.
10. If using interrupts, ensure that GIE and PEIE (bits 7 and 6) of the INTCON register are set.

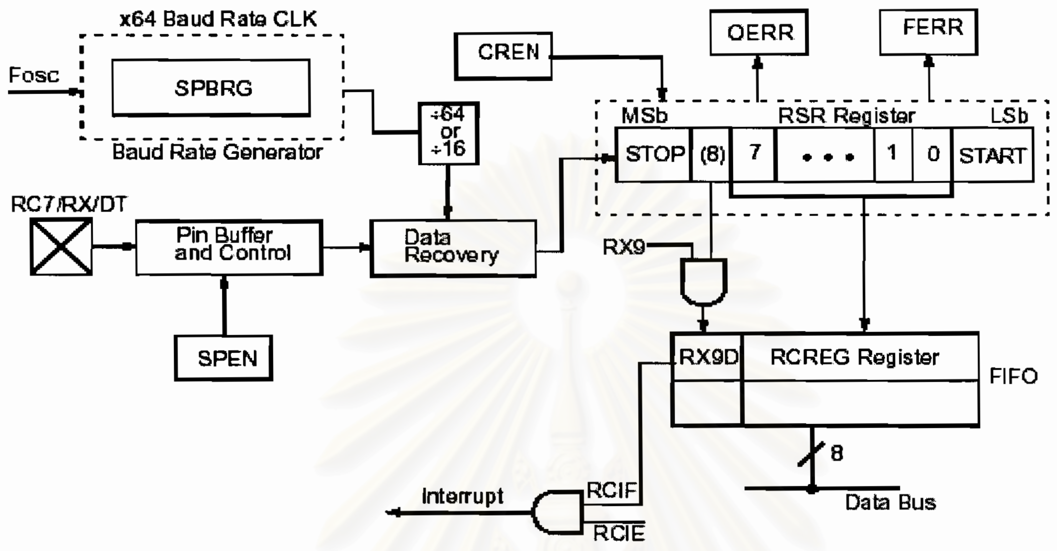


Fig 15-A. USART Receive block diagram

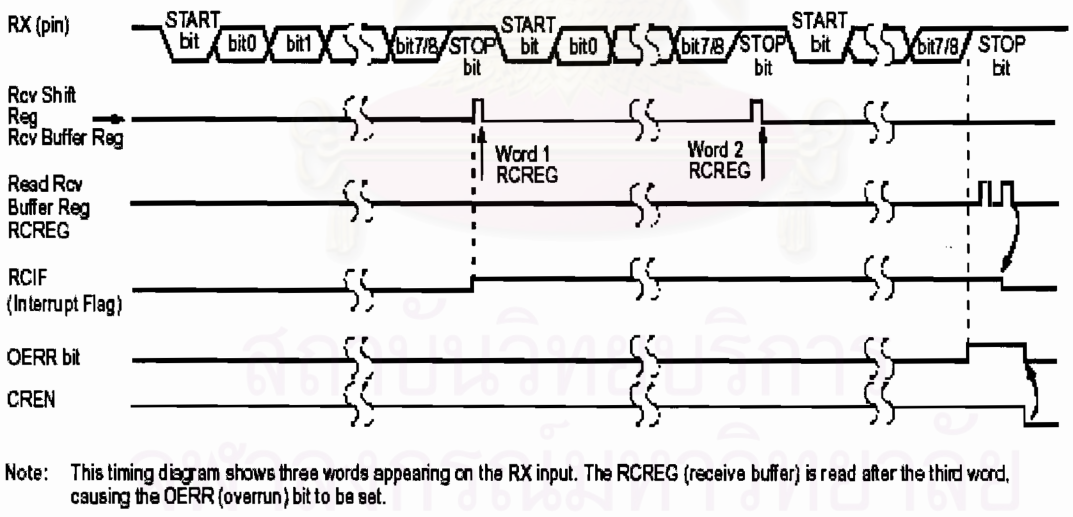

RX (pin)
START bit
bit0
bit1
bit7/8
STOP bit
Rcv Shift Reg → Rcv Buffer Reg
Word 1 RCREG
Word 2 RCREG
Read Rcv Buffer Reg RCREG
RCIF (Interrupt Flag)
OERR bit
CREN
Note: This timing diagram shows three words appearing on the RX input. The RCREG (receive buffer) is read after the third word, causing the OERR (overrun) bit to be set.

Fig 15-B. Asynchronous Reception

Fig 15-A and Fig 15-B show the USART Receive block diagram and Asynchronous Reception respectively. TABLE V shows Register Associated with Asynchronous Reception.

TABLE V
REGISTER ASSOCIATED WITH ASYNCHRONOUS RECEPTION

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF[1]	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
1Ah	RCREG	USART Receive Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE[1]	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown. - = unimplemented locations read as '0'. Shaded cells are not used for asynchronous reception.
Note 1: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.

Analog-to-Digital Converter: PIC16F877 has eight inputs ADC that are used to read any analog data that come from sensor board. This ADC uses successive approximation technique to achieve a very low acquisition time (Analog to Digital conversion time). The A/D conversion of the analog input signal results in a corresponding 10-bit digital number. By using the multiplexage technique, we can connect all these 8 analog inputs to only one A/D module. This can be done by selecting the ADCON0, bit 5 to 3. See ADCON0 register in Fig 16 and Fig 17 for more detail.

The A/D module has high and low voltage reference input that is software selectable to some combination of VDD, VSS, RA2, or RA3. The A/D converter has a unique feature of being able to operate while the device is in SLEEP mode. To operate in SLEEP, the A/D clock must be derived from the A/D's internal RC oscillator however; we prefer to use Fosc/32 because at that time our CPU module is not operating at sleep mode any more. Wake up from low-power mode is not activated from any interrupt (Timer interrupt or Peripheral interrupt) but from our program itself. See Fig 16.

The A/D module has four registers. These registers are:

- A/D Result High Register (ADRESH)
- A/D Result Low Register (ADRESL)
- A/D Control Register0 (ADCON0)
- A/D Control Register1 (ADCON1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

bit 7-6 **ADCS1:ADCS0**: A/D Conversion Clock Select bits
00 = FOSC/2
01 = FOSC/8
10 = FOSC/32
11 = FRC (clock derived from the internal A/D module RC oscillator)

bit 5-3 **CHS2:CHS0**: Analog Channel Select bits
000 = channel 0, (RA0/AN0)
001 = channel 1, (RA1/AN1)
010 = channel 2, (RA2/AN2)
011 = channel 3, (RA3/AN3)
100 = channel 4, (RA5/AN4)
101 = channel 5, (RE0/AN5)(1)
110 = channel 6, (RE1/AN6)(1)
111 = channel 7, (RE2/AN7)(1)

bit 2 **GO/$\overline{\text{DONE}}$**: A/D Conversion Status bit
If ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion)
0 = A/D conversion not in progress (this bit is automatically cleared by hardware when the A/D conversion is complete)

bit 1 **Unimplemented**: Read as '0'

bit 0 **ADON**: A/D On bit
1 = A/D converter module is operating
0 = A/D converter module is shut-off and consumes no operating current

Note 1: These channels are not available on PIC16F873/876 devices.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Fig 16. ADCON0 Register (Address: 1Fh)

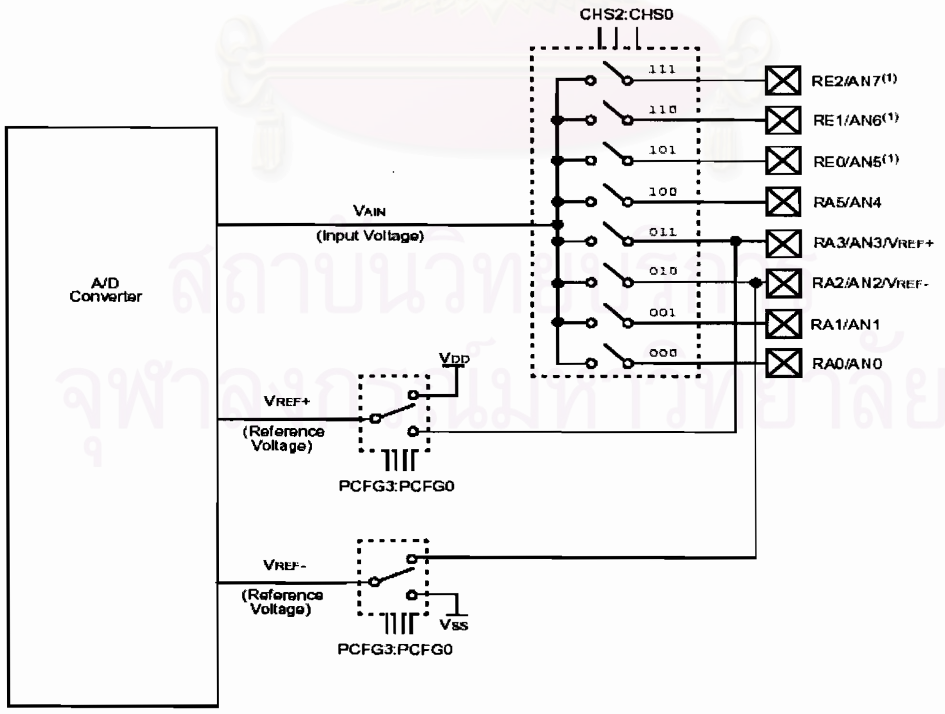
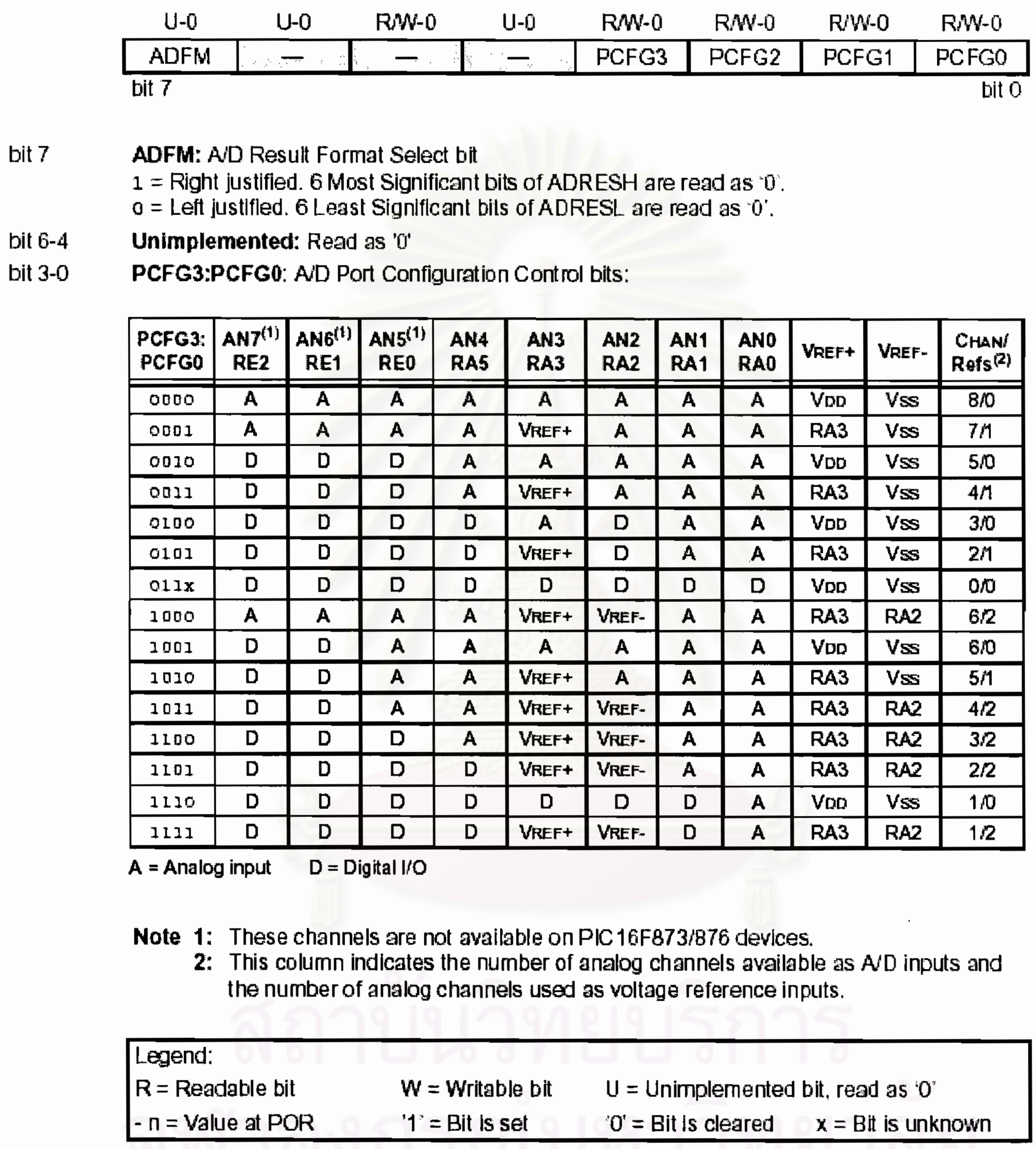


Fig 17. A/D Block Diagram

The ADCON0 register controls the operation of the A/D module. The ADCON1 register, shown in Fig 18, configures the functions of the port pins. The port pins can be configured as analog inputs (RA3 can also be the voltage reference), or as digital I/O. The Registers/Bits associated with A/D is shown in TABLE VI.



U-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7	**ADFM:** A/D Result Format Select bit 1 = Right justified. 6 Most Significant bits of ADRESH are read as '0'. 0 = Left justified. 6 Least Significant bits of ADRESL are read as '0'.
bit 6-4	**Unimplemented:** Read as '0'
bit 3-0	**PCFG3:PCFG0:** A/D Port Configuration Control bits:

PCFG3: PCFG0	AN7[(1)] RE2	AN6[(1)] RE1	AN5[(1)] RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	VREF+	VREF-	CHAN/ Refs[(2)]
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	RA3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	RA3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	RA3	VSS	2/1
011x	D	D	D	D	D	D	D	D	VDD	VSS	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	RA3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

A = Analog input D = Digital I/O

Note 1: These channels are not available on PIC16F873/876 devices.
2: This column indicates the number of analog channels available as A/D inputs and the number of analog channels used as voltage reference inputs.

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared x = Bit is unknown

Fig 18. ADCON1 Register (Address: 9Fh)

The following steps should be followed for doing A/D conversion:

1. Configure the A/D module:
   - Configure analog pins/voltage reference and digital I/O (ADCON1)

- Select A/D input channel (ADCON0)
- Select A/D conversion clock (ADCON0)
- Turn on A/D module (ADCON0)

2. Configure A/D interrupt (if desired):
    - Clear ADIF bit
    - Set ADIE bit
    - Set PEIE bit
    - Set GIE bit
3. Wait the required acquisition time.
4. Start conversion:
    - Set GO/ $\overline{\text{DONE}}$ bit (ADCON0)
5. Wait for A/D conversion to complete, by either:
    - Polling for the GO/$\overline{\text{DONE}}$ bit to be cleared (with interrupts enabled); OR
    - Waiting for the A/D interrupt
6. Read A/D result register pair (ADRESH: ADRESL), clear bit ADIF if required.
7. For the next conversion, go to step 1 or step 2, as required. The A/D conversion time per bit is defined as T_{AD}. A minimum wait of $2T_{AD}$ is required before the next acquisition starts.

TABLE VI
REGISTERS/BITS ASSOCIATED WITH A/D

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on $\overline{\text{MCLR}}$, WDT
0Bh,8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF[1]	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE[1]	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	uuuu uuuu
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	uuuu uuuu
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON	0000 00-0	0000 00-0
9Fh	ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0	--0- 0000	--0- 0000
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	--11 1111
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	--0u 0000
89h[1]	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	0000 -111
09h[1]	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	---- -uuu

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used for A/D conversion.

In order, for A/D converter, to meet the specified accuracy, the charge holding capacitor mush be allowed to fully charge to the input channel voltage level. The maximum recommended impedance for analog source is 10 kΩ. As the impedance is decrease, the acquisition time may be deceased, we prefer to user an amplifier for any analog source which have high impedance output (to achieve low impedance output for short acquisition time). According to the datasheet of PIC16F877, the minimum acquisition time is about 19.72 μs. After the analog input channel is selected, this acquisition time must be done before the conversion can be started.

The information about the minimum acquisition time is, actually, not really important for us, however it can give us idea to estimate how long A/D converter will take for each conversion. Why this information is not so useful for us? According to the A/D conversion steps, in step 5, we can check $GO/\overline{DONE}$ bit to be cleared. As this bit is automatically cleared by hardware when the A/D conversion is complete, we can know when the A/D converter is finish and so, we can start to do other tasks.

In fact, the newer version of PICBaic Pro Compiler have a command ADCIN to do this job but our compiler does not support this command, so we have to access to the register and follow all that 7 steps to get A/D conversion result. The similar reason is used for USART Asynchronous reception.

I^{2}C Bus: PIC16F87X has Master Synchronous Serial Port (MSSP) module build in. This serial interface is useful for communicating with other peripheral or microcontroller devices. These peripheral devices may be serial EEPROMs, shift registers, display drivers, A/D converters, etc. The MSSP module can operate in one of two modes:

- Serial Peripheral Interface (SPI)
- Inter-Integrated Circuit (I^{2}C)

However, we interest only I^{2}C mode. Fig 19-A and Fig 19-B show the block diagrams for the two different I^{2}C modes of operation. In our application, I^{2}C Master Mode is used.

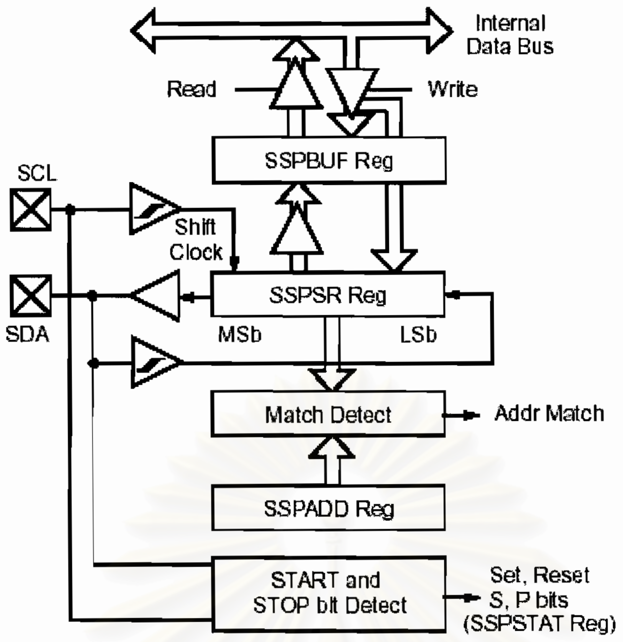


Fig 19-A. I^{2}C Slave Mode Block Diagram

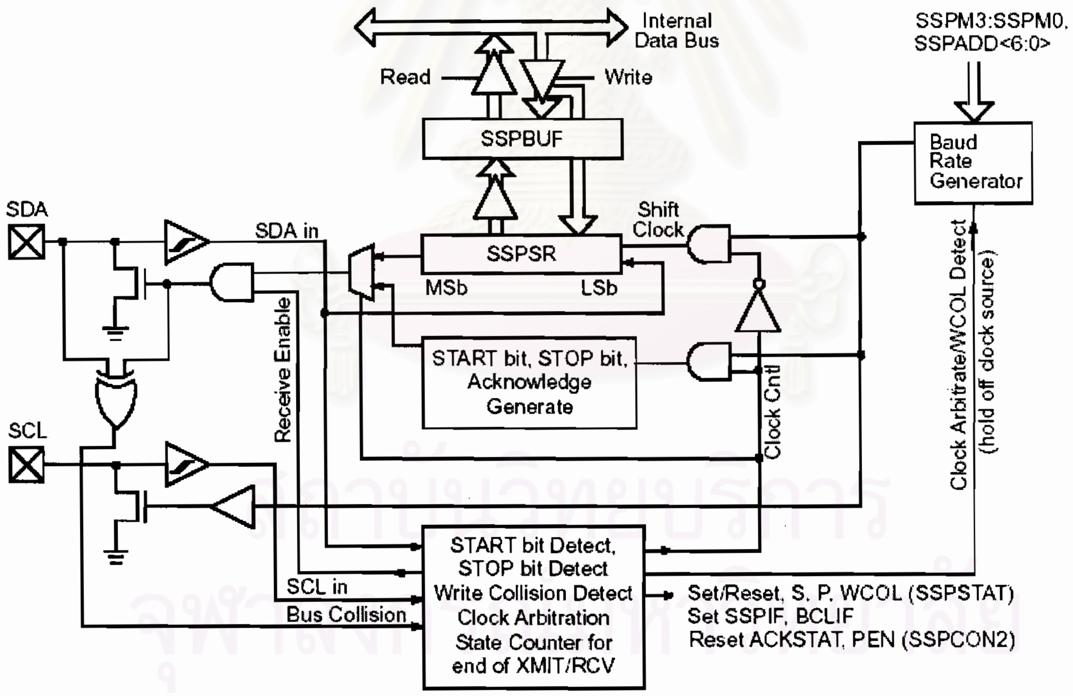

Internal Data Bus
SSPM3:SSPM0, SSPADD<6:0>
Read
Write
SSPBUF
Baud Rate Generator
SDA
SDA in
Shift Clock
SSPSR
MSb
LSb
Receive Enable
START bit, STOP bit, Acknowledge Generate
Clock Cntl
Clock Arbitrate/WCOL Detect (hold off clock source)
SCL
START bit Detect, STOP bit Detect Write Collision Detect Clock Arbitration State Counter for end of XMIT/RCV
SCL in
Bus Collision
Set/Reset, S, P, WCOL (SSPSTAT)
Set SSPIF, BCLIF
Reset ACKSTAT, PEN (SSPCON2)

Fig 19-B. I^{2}C Master Mode Block Diagram

As this job is supported by our PICBasic Pro Compiler, we would end the detail here, no go in deep into the register that associated with I^{2}C operation. However, in section 3.2.1, we will talk about I^{2}C bus protocol characteristic in more detail.

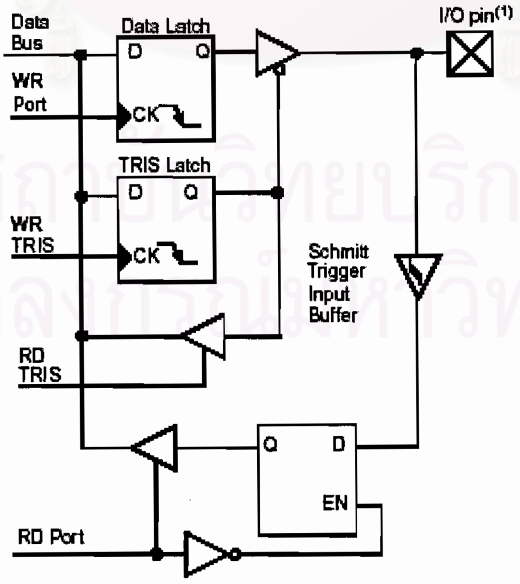
I/O Port: Generally, some pins for these I/O ports are multiplexed with an alternate function for the peripheral features on the device such as PORTC that is alternate with USART and I^{2}C Bus. In general, when a peripheral is enabled, that pin may not be used as a general purpose I/O pin. In this project, PORTD is defined as input port not parallel slave port even though it supports this alternate function.

PORTD is an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configurable as an input or output. It can be configured as an 8-bit wide microprocessor port (parallel slave port) by setting control bit PSPMODE (TRISE<4>). In this mode, the input buffers are TTL. See TABLE VII for the registers associated with PORTD and Fig 20 for PORTD block diagram in I/O port mode.

TABLE VII
REGISTERS ASSOCIATED WITH PORTD

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
08h	PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx xxxx	uuuu uuuu
88h	TRISD	PORTD Data Direction Register								1111 1111	1111 1111
89h	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction Bits			0000 -111	0000 -111

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by PORTD.



Note 1: I/O pins have protection diodes to VDD and VSS.

Fig 20. PORTD Block Diagram in I/O Port Mode

Setting a TRISD bit (=1) will make the corresponding PORTD pin an input (i.e., put the corresponding output driver in a hi-impedance mode). Clearing a TRISD bit (= 0) will make the corresponding PORTA pin an output (i.e., put the contents of the output latch on the selected pin).

Reading the PORTD register reads the status of the pins, whereas writing to it will write to the port latch. All write operations are read-modify-write operations. Therefore, a write to a port implies that the port pins are read; the value is modified and then written to the port data latch.

So, by setting TRISD=%11111111, PORTD is configured as 8-bit input and then we can read the PORTD register directly (i.e., read only one time but get the value (status) of all pins).

Note that all the information here are copied from Microchip datasheet and modified for better understanding. It does not cover all the information but enough for this project. Some information is excluded because we do not use it while doing our project. We refer only what we have used.

3.1.2 Electrically Erasable and Programmable Read Only Memory (EEPROM)

The AT24C512 is used to store the data of surrounding environment and other option as describe in Chapter 2. It provides 524,288 bits of serial EEPROM organized as 65,536 words of 8 bits each. The device's cascadable feature allows up to 4 devices to share a common 2-wire bus. This can be done by selecting the corresponding hardware address (A_1 and A_0).

This device is optimized for use in many industrial and commercial applications where low-power and low-voltage operations are essential. It can operate at very low-voltage (down to 1.8V). It uses 2-wire serial interface (I^2C in short) to achieve bi-directional data transfer protocol and can function up to 1 MHz clock at 5V power supply. It also has Schmitt Triggers to suppress the inputs noise and other feature; however, the most important point for us is the Self-timed Write Cycle, which determines the speed of scanning process in our project. In general, this Self-timed

Write Cycle is about 10 ms but, with this device, we can achieve a very low Self-timed Write Cycle (5 ms Max).

According to our experience, some applications may need the high speed of clock for a very short reading process such as Matrix Display Board whereas this application, low Self-timed Write Cycle is needed because we use writing process more often then reading process. One more thing, we can not use page write to write the data into the EEPROM because our data length is not fix, so it will cause a serious problem while writing across the page, therefore, Byte write is preferred and Self-timed Write Cycle become the most important factor. Fig 21 shows the EEPROM Block Diagram. Fig 22-A and Fig 22-B show the Random Read and Sequential Read process whereas Fig 23-A and Fig 23-B show the Byte write and Page write process. For the Bus characteristics, we will talk about it in section 3.2.1.

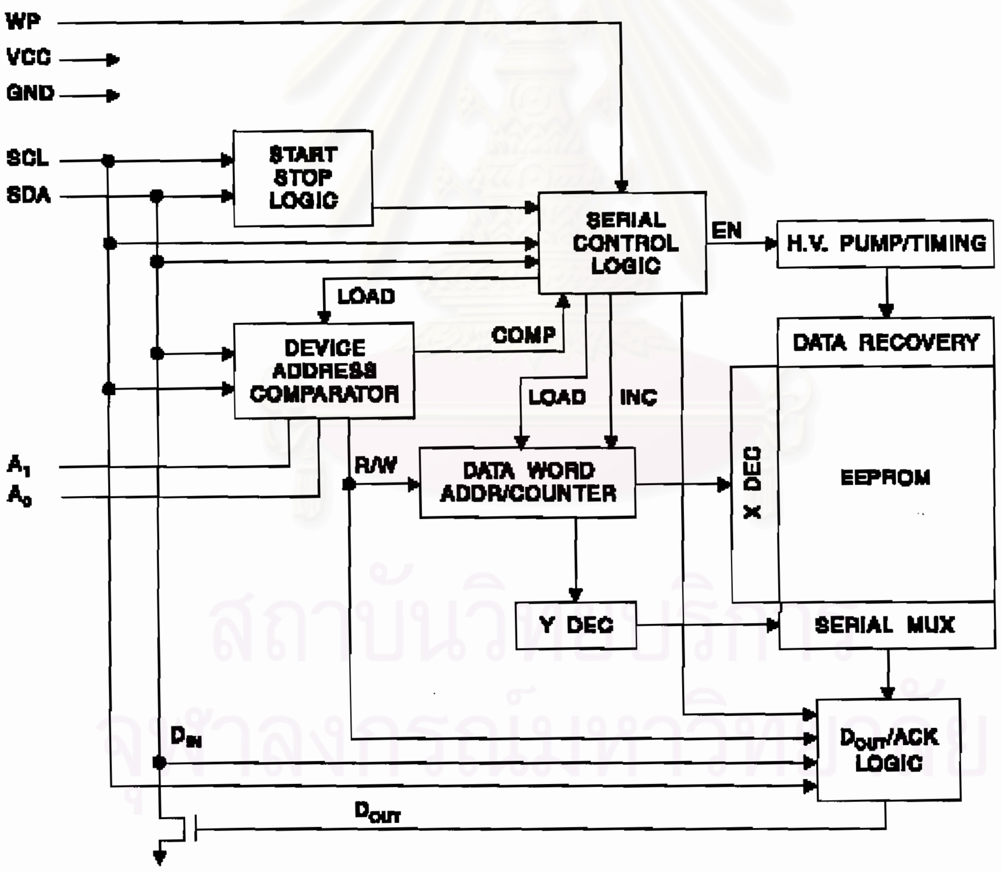


Fig 21. AT24C512 Block Diagram

Serial Clock (SCL): This input is used to synchronize the data transfer from and to the device.

Serial Data (SDA): This is a bi-directional pin used to transfer addresses and data into and data out of the device. It is and open drain terminal, therefore, the SDA bus requires a pull-up resistor to Vcc (typical 10 kΩ for 100 kHz, 2 kΩ for 400 kHz and 1 MHz).

Device/Addresses (A_0, A_1): These inputs are used for multiple device operations. The levels on these inputs are compared with the corresponding bits in the slave address. The chip is selected if the compare is true. With these two addresses, we can connect up to 2^2 = 4 devices on the same bus.

A Read operation requires two 8-bit data word addresses following the device address word and acknowledgment (the read/write select bit in the device address word is set to one). In principle, there are three read operations: current address read, random address read and sequential read however, we interest only random address read and sequential read because we only use these two modes in our project.

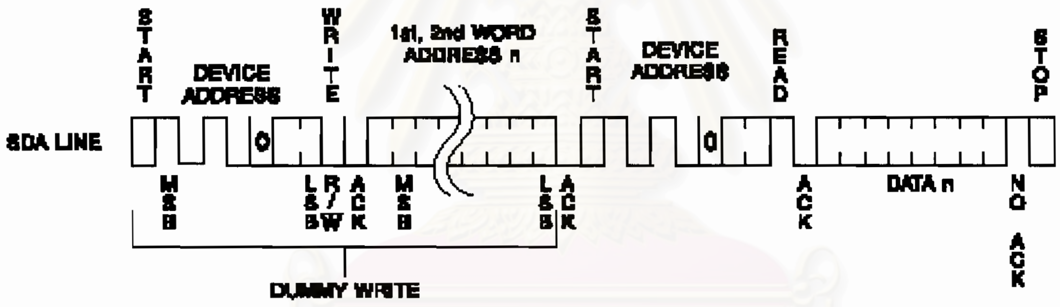


Fig 22-A. Random Read

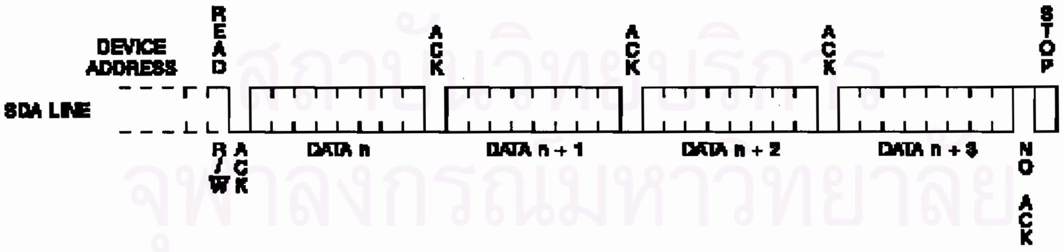


Fig 22-B. Sequential Read

A write operation is divided into two modes:

Byte Write: A write operation requires two 8-bit data word addresses following the device address word and acknowledgment. Upon receipt of this address, the EEPROM will again respond with a zero and then clock in the first 8-bit data word. Following receipt of the 8-bit data word, the EEPROM will output a zero. The addressing device, such as a microcontroller, then must terminate the write sequence with a stop condition. At this time the EEPROM enters an internally timed write cycle to the nonvolatile memory. All inputs are disabled during this write cycle and the EEPROM will not respond until the write is complete (refer to Fig 23-A).

Page Write: The 512K EEPROM is capable of 128-byte page writes. A page write is initiated the same way as a byte write, but the microcontroller does not send a stop condition after the first data word is clocked in. Instead, after the EEPROM acknowledges receipt of the first data word, the microcontroller can transmit up to 127 more data words. The EEPROM will respond with a zero after each data word received. The microcontroller must terminate the page write sequence with a stop condition (refer to Fig 23-B).

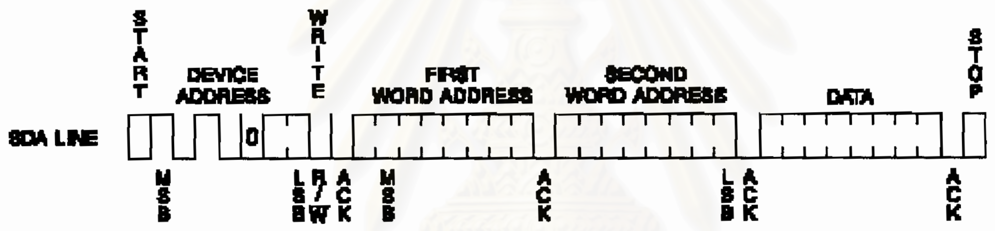


Fig 23-A. Byte Write

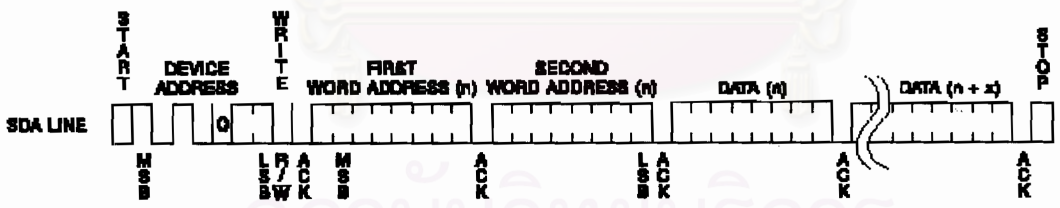


Fig 23-B. Page Write

3.1.3 Real Time Clock (RTC)

There are so many RTC's available on the market today; however, DS1307 was chosen to be our real time clock because it has a programmable square wave output signal and independence of power supply frequency and has backup battery. In fact, this is not the best one but enough for our project. TABLE IIX shows the best RTC's from DALLAS Semiconductor Company.

TABLE IIX
BEST RTC OF DALLAS SEMICONDUCTOR COMPANY

Part	Supply Voltage (V)	Backup Supply Input	Trickle Charger	Time of Day Alarm(s)	Square-Wave Output	Watchdog	Reset Output	SRAM
DS1338	1.8, 3.0, 3.3	✔			Programmable			56 x 8
DS1374	1.8, 3.0, 3.3, 5	✔	✔	✔	Programmable	✔	✔	
DS1307	5.0	✔			Programmable			56 x 8
DS1337	1.3 to 5.0			✔	Programmable			
DS1339	2.0, 3.0, 3.3	✔	✔	✔	Programmable			
DS1371	1.3 to 5.5			✔	Programmable	✔		
DS1672	2.0, 3.0, 3.3	✔	✔				✔	
MAX6900	2.0 to 5.5							31 x 8

The DS1307 is a low power Serial Real Time Clock, full Binary-Coded Decimal (BCD) clock/calendar plus 56 bytes of nonvolatile SRAM. Address and data are transferred serially via a 2-wire, bi-directional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power sense circuit that detects power failures and automatically switches to the battery supply. A lithium battery with 48 mAH or greater will back up the DS1307 for more then 10 years in the absence of power at 25°C.

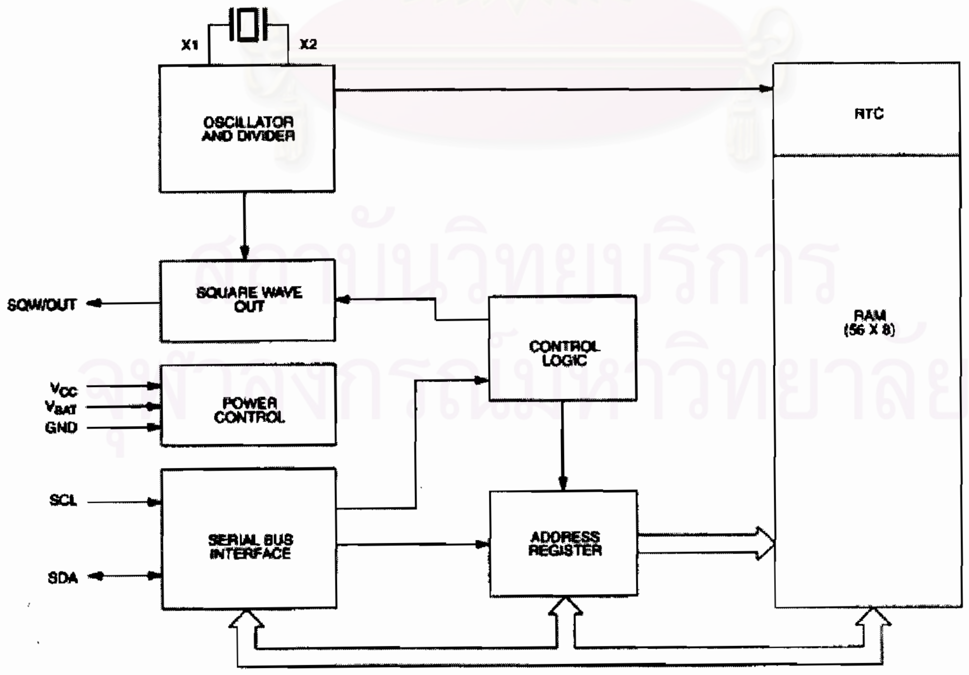


Fig 24. RTC Block Diagram

The block diagram in Fig 24 shows the main elements of the Serial Real Time Clock. It operates as a slave device on the serial bus (I^2C). It supports Slave Receiver Mode (Data Write) and Slave Transmitter Mode (Data Read). See Fig 25-A and Fig 25-B for more detail.

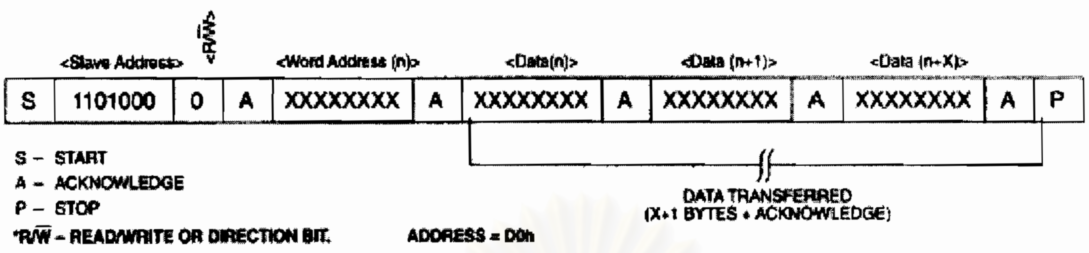


Fig 25-A. Slave Receiver Mode

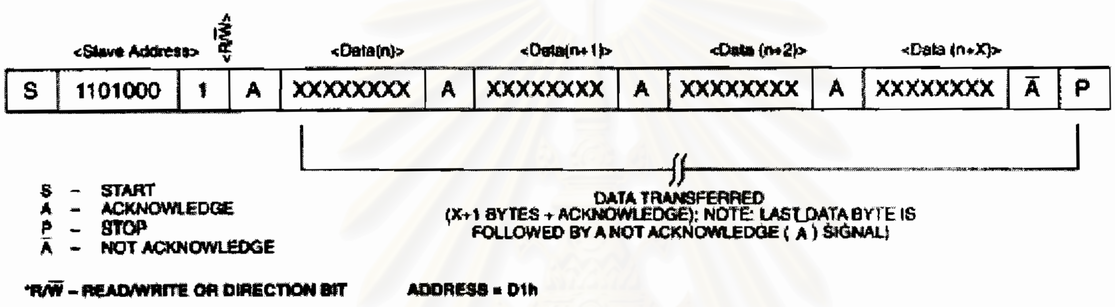


Fig 25-B. Slave Transmitter Mode

The time and calendar information is obtained by reading the appropriate register bytes. The real time clock registers are illustrated in Fig 26. The time and calendar are set or initialized by writing the appropriate register bytes. As the contents of the time and calendar registers are in the BCD format, the conversion is needed in order to get the real data for time and calendar.

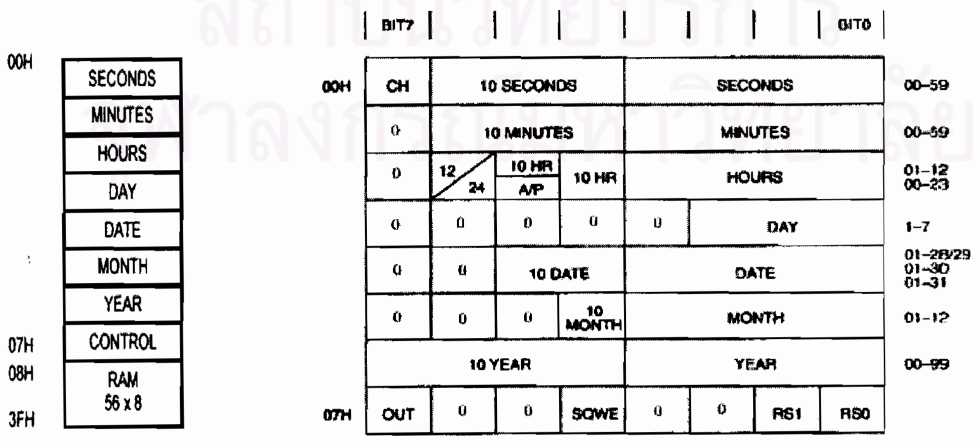


Fig 26. Address Map and Timekeeper Registers

SQW/OUT (Square Wave/Output Driver): When enable, the SQWE bit set to 1, the SQW/OUT pin outputs one of four square waves (see Table IX), however, 1 Hz square wave is chosen for our application.

TABLE IX
SQUARE WAVE OUTPUT FREQUENCY

RS1	RS0	SQW OUTPUT FREQUENCY
0	0	1Hz
0	1	4.096kHz
1	0	8.192kHz
1	1	32.768kHz

3.1.4 Digital to Analog Converter (DAC)

From the Internet, we can find so many available integrated circuits that assure the digital to analog conversion such as AD5301, AD5311 and AD5321, from Analog Devices Company, are single 8-bit, 10-bit, and 12-bit buffered voltage-output DACs that operate from a single 2.5 V to 5.5 V supply, consuming 120 μA at 3 V. However, PCF8591 is chosen because it is available in Thailand.

The PCF8591 is a single-chip product from Philips Semiconductor, single-supply low power 8-bit CMOS data acquisition device with four analog inputs, one analog output and a serial I2C-bus interface. Three address pins A0, A1 and A2 are used for programming the hardware address, allowing the use of up to eight devices connected to the I2C-bus without additional hardware. Address, control and data to and from the device are transferred serially via the two-line bidirectional I2C-bus.

The functions of the device include analog input multiplexing, on-chip track and hold function, 8-bit analog-to-digital conversion and an 8-bit digital-to-analog conversion. The maximum conversion rate is given by the maximum speed of the I2C-bus but this factor is not important for our application because this time delay is very small comparing to the SMS process (reading PDU-SMS, convert to Text-SMS and run script execution function).

Even though PCF8591 support A/D conversion, we interest only D/A conversion. To achieve two analog-output ports, two pieces of PCF8591 are needed. Fig 27 shows the block diagram of this IC.

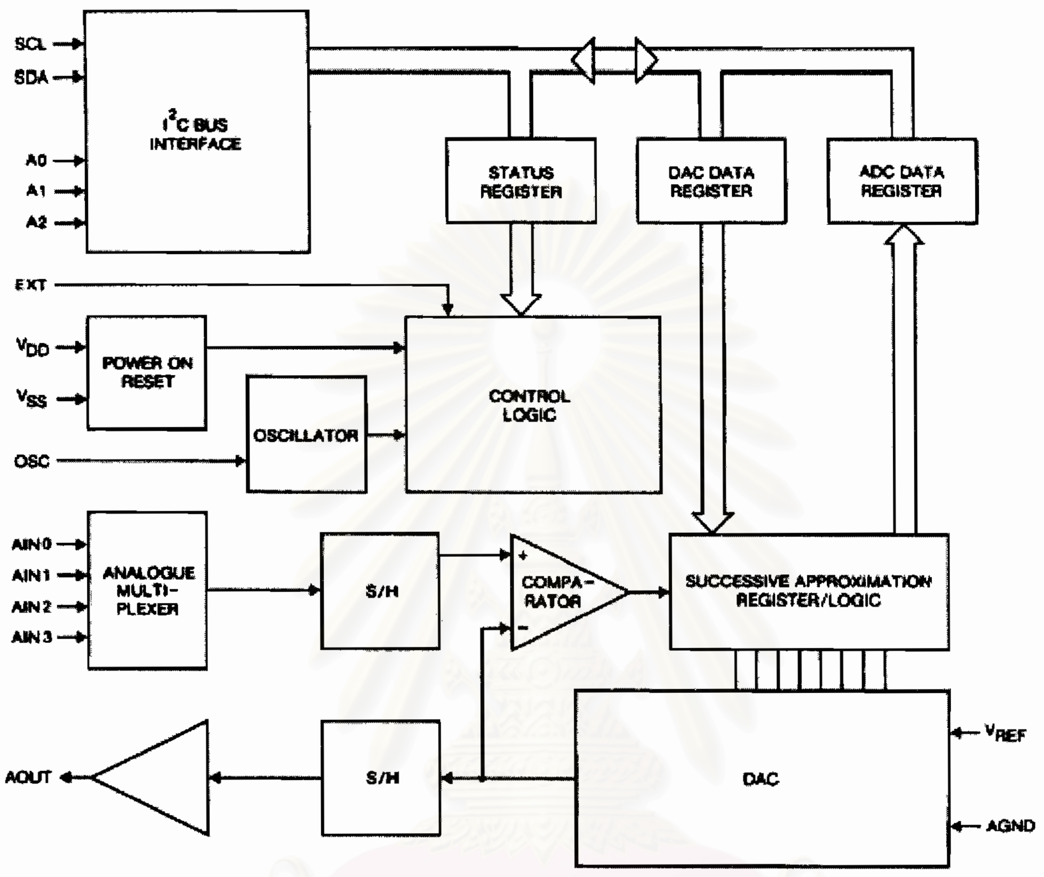


Fig 27. PCF8591 Block Diagram

The format of writing the value to analog output port is shows in Fig 28. This format is for continuous writing process, it means that after select the corresponding I^{2}C device, we can sent the digital value to this converter one by one and continue until the stop condition is generated. For us, we only send the analog value just one time when we get a command via SMS from the central computer or manager.

The third byte sent to a PCF8591 device is stored in the DAC data register and is converted to the corresponding analog voltage using the on-chip D/A converter. This D/A converter consist of a resistor divider chain connected to the external reference voltage with 256 taps and selection switches. The tap-decoder switches one of these taps to the DAC output line (see Fig 29).

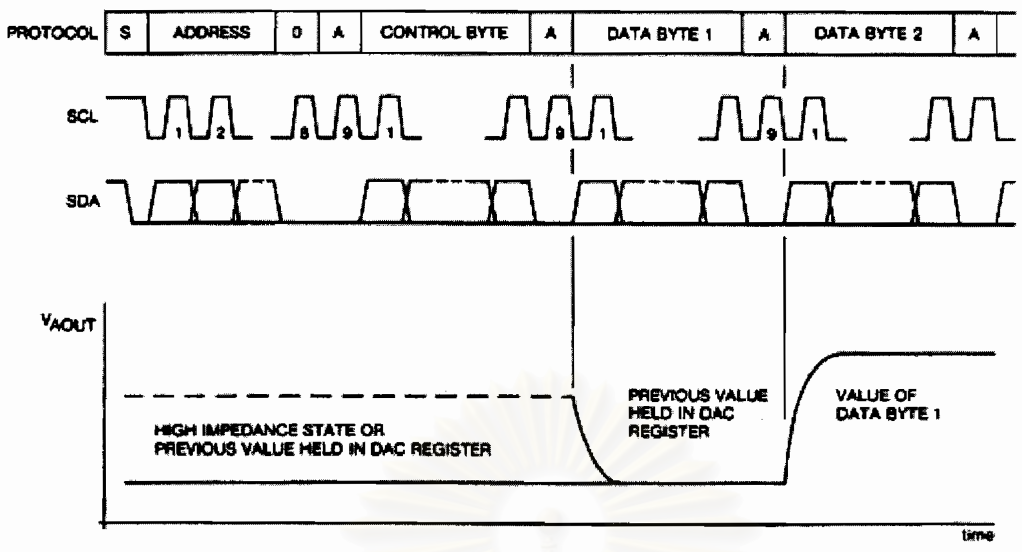


Fig 28. D/A Conversion Sequence

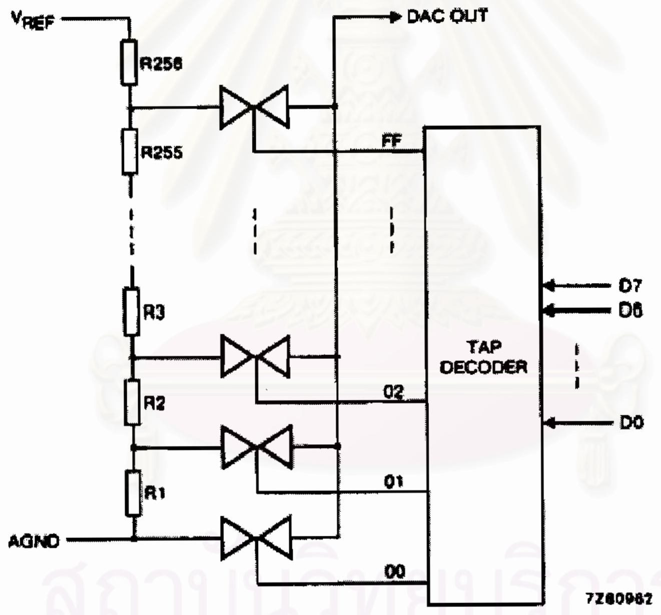


Fig 29. DAC Resistor Divider Chain

With PICBasic Pro Compiler, this process can be done just with a standard command I2CWRITE with 8-bit address. For PCD8591, Address is a device address whereas control byte is a word address in standard 8-bit I^2C Byte/Page writes. The upper nibble of the control register is used for enabling the analog output, and for programming the analog inputs as single-ended or differential inputs. If the auto-increment flag is set the channel number is incremented automatically after each

A/D conversion (See Fig 30). So for our application, the control byte can be set to %01000000=64 or %01000100=68.

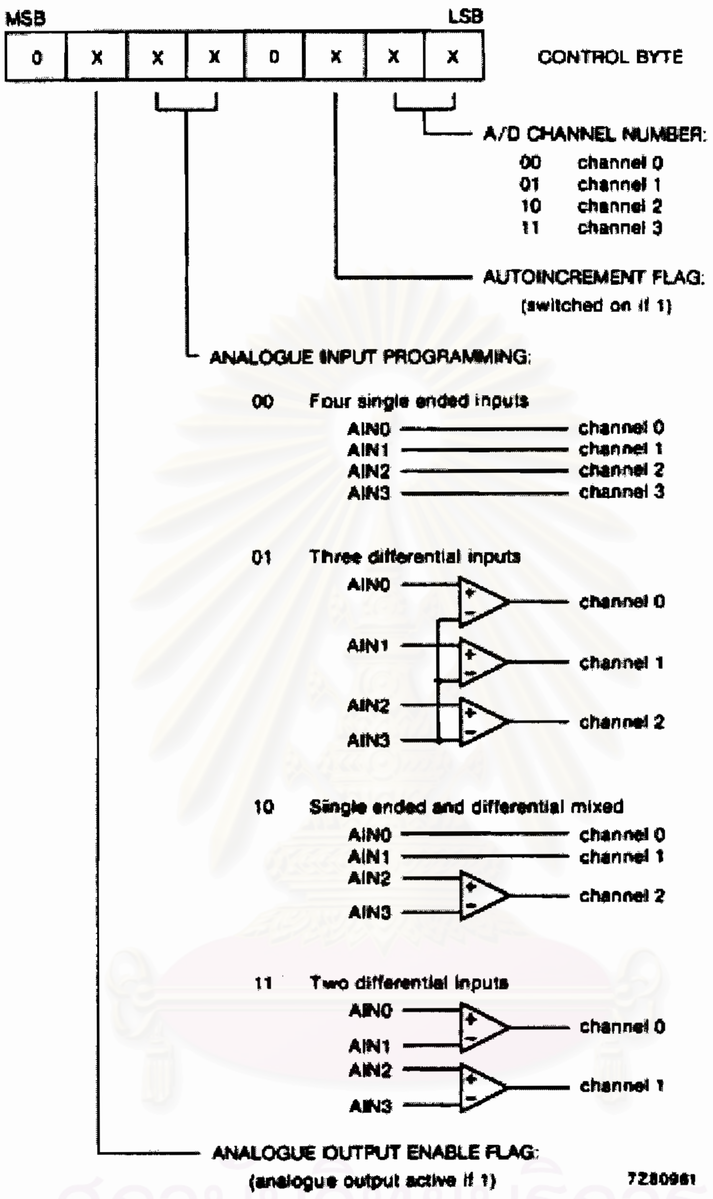


Fig 30. PCF8591 Control Byte Register

3.1.5 PCF8574-Remote I/O Expander

The PCF8574 is silicon CMOS circuit. It provides general-purpose remote I/O expansion for most microcontroller families via the two-line bidirectional bus (I^2C). The device consists of an 8-bit quasi-bidirectional port and an I^2C-bus interface. The PCF8574 has a low current consumption and includes latched outputs with high current drive capability for directly driving LEDs. It also possesses an interrupt line

(INT), which can be connected to the interrupt logic of the microcontroller. By sending an interrupt signal on this line, the remote I/O can inform the microcontroller if there is incoming data on its ports without having to communicate via the I^2C-bus. This means that the PCF8574 can remain a simple slave device, however, as our project needs to read and store data every scanning time, this feature seem to be meaningless for us. Fig 31 shows the PCF8574 block diagram.

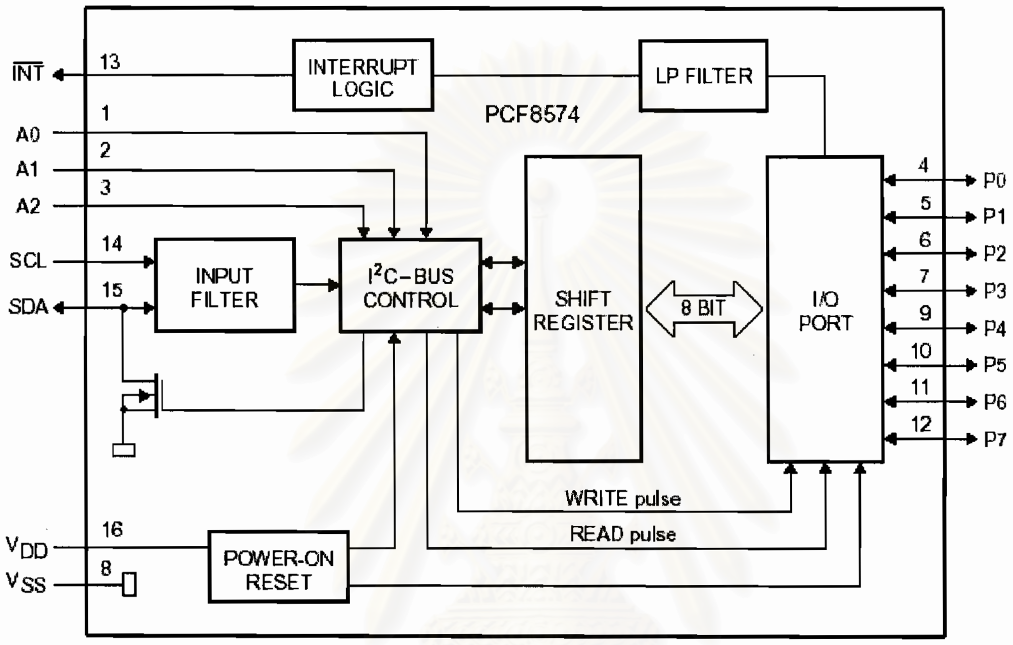


Fig 31. PCF8574 Block Diagram

The PCF8574 and PCF8574A versions differ only in their slave address as shown in Fig 32.

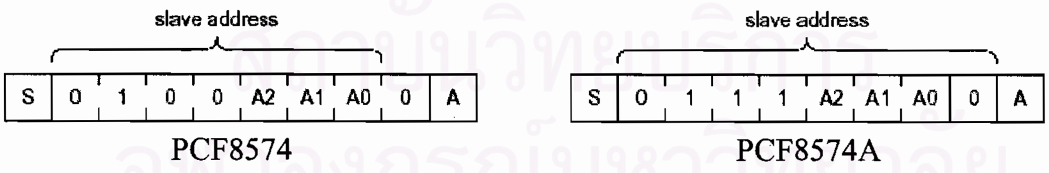


Fig 32. PCF8574 and PCF8574A Slave Address

Each of the PCF8574's eight I/Os can be independently used as an input or output. Input data is transferred from the port to the microcontroller by the READ mode. Output data is transmitted to the port by the WRITE mode (see Fig 33). In this project, it is used PCF8574A as an output port only therefore; we don't show READ mode chronogram.

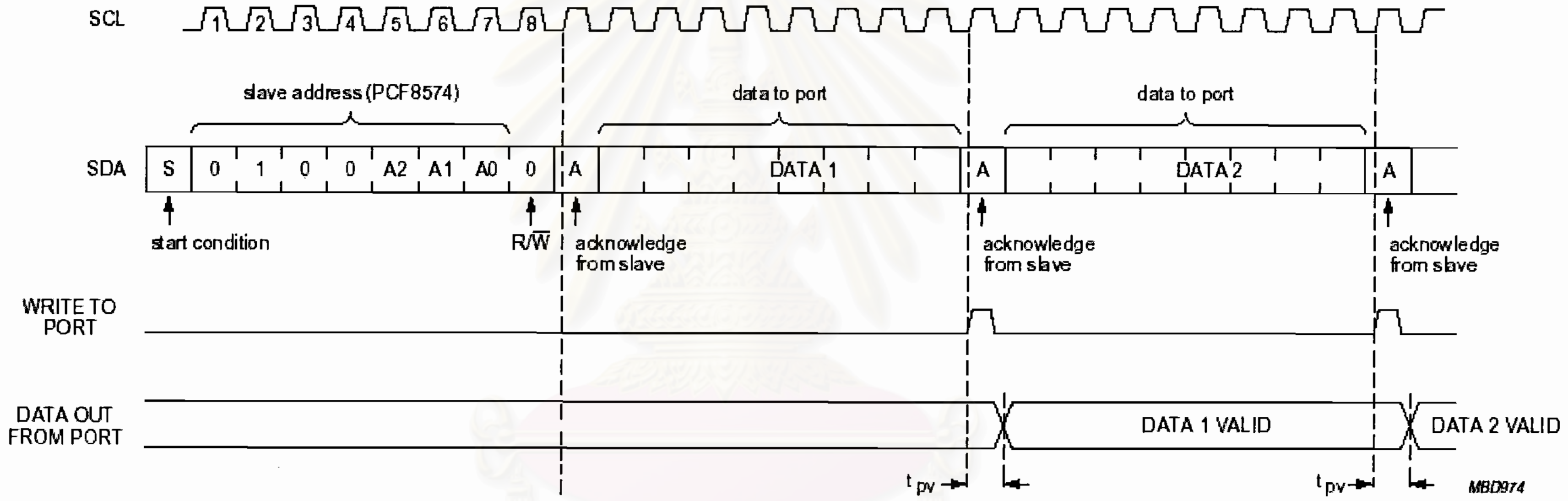


Fig 33. Write Mode (Output)

According to write mode wave form, we can see that it is not a standard format for I2COUT command; however, by setting the 8-bit address field by the data, this standard command still can use prettily.

3.1.6 GSM Module

Besides using ERICSSON T68 as wireless link for the system, an alternate solution is founded. The GM862-GPRS is a GSM GPRS module dual band GSM 900/1800 with Easy GPRS Embedded (TCP/IP stack inside). The GM862-GSM wireless data modules are the ready solution for all M2M wireless applications.

The GM862-GSM is specifically designed and developed by Telit for OEM usage and dedicated to cost effective voice and telematics applications where the Packed Data features of the GPRS are not constrain such as:

- Telemetry and Telecontrol (SCADA applications)
- Security systems
- Cost Effective Vending machines
- Low cost POS terminals
- Phones and Payphones
- Automotive and Fleet Management applications
- Domestic Appliances with simple remote control

It supports:

- E-GSM 900/1800 MHz
- GSM: Compliant with GSM Phase 2/2+
- Output power up to 2W for Class 4 at GSM 900 MHz and 1W for Class 1 at GSM 1800 MHz
- Control via AT commands (ITU, GSM, GPRS and Telit Supplementary)
- Supply Voltage: 3.4 to 4.2V, Nominal: 3.8V
- Power consumption: Idle mode: < 3.5 mA
- Dedicated mode 250 mA (average)

For more information on Telit Globalstar telecommunication modules, see www.telital.com or www.GM862.com or www.roundsolution.com. Fig 34 shows the overview of GM862-GPRS module.

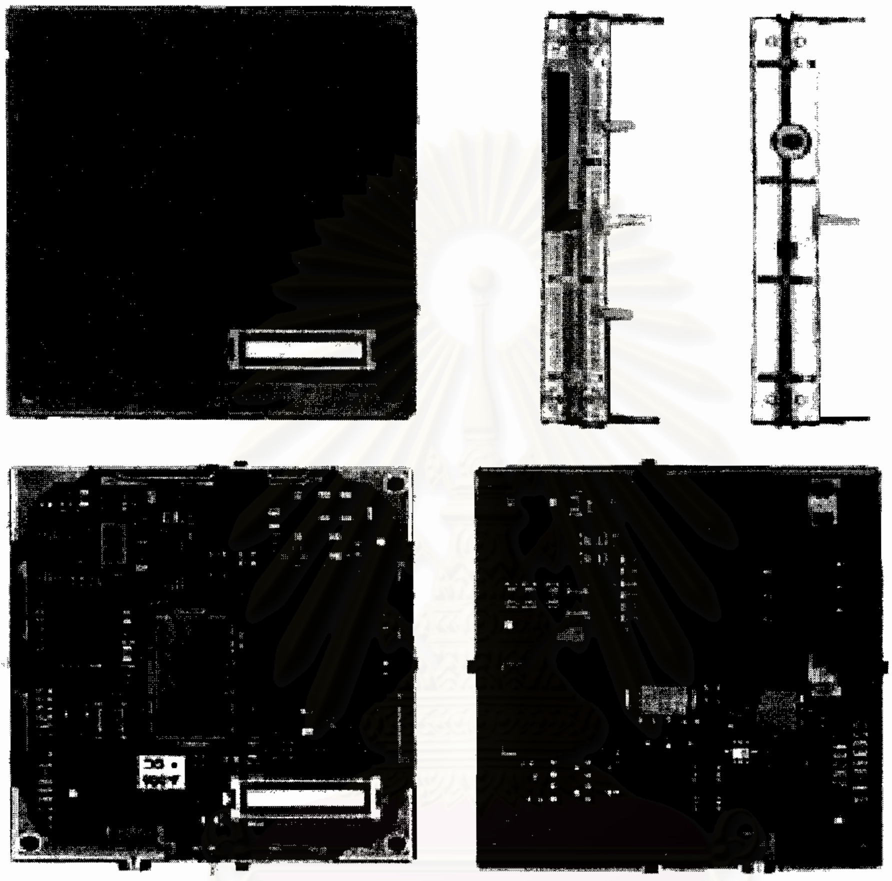


Fig 34. GM862-GPRS Overview

Even though GM862-GSM is enhanced with the on board SIM Reader and can support so many features such as GPRS, Voice, Circuit Switch Data Transfer, Fax, Phonebook and SMS support. However, we only use some part of these features, Circuit Switch Data Transfer and SMS, the other will not take into account.

As it is supplied with 3.8V, a power supply circuitry must be taken into account very carefully. This part is the most important part in the full product design and they strongly reflect on the product overall performances. Fortunately, Round Solution has developed an extension board for this GSM module.

The RS-ES-S1B board is developed by Round Solution to help us to get start with GM862-GPS. It has:

- 50 pin Molex connector to interface the GM862 directly
- 9 pin RS232 interface
- Jumper to power on the RS232 level shifter
- 30-pin connector to support ON/OFF, AXE, Reset, GPIO, STAT-LED, audio and serial port with full handshaking.
- ZIF connector to CMOS camera

TABLE X shows the pin-out of 30-pin connector. As we can see from this TABLE, 30-pin connector support many important connections between GM862 and RS-ES-SB1, however, we interest only in $\overline{\text{ON}}$ pin and START pin. +3.7V and GND is internally connected, no need to take care about this. Thank Round Solution Company for developing this useful starting board.

TABLE X
PIN-OUT OF 30-PIN INTERFACE CONNECTOR

Pin Number	**Name**	**Pin Number**	**Name**
1	Ear HF+	**16**	**START_LED**
2	Ear HT-	**17**	**GND**
3	Ear HF-	18	Rx_PROG
4	Ear HT+	19	CTS
5	**GND**	20	RING
6	Mic HF-	21	DSR
7	Mic HT+	22	DCD
8	Mic HF+	23	Tx_PROG
9	Mic HT-	24	DTR
10	$\overline{\text{ON}}$	25	RTS
11	**START**	**26**	**GND**
12	$\overline{\text{RESET}}$	**27**	**PWR-GND**
13	GPIO2	**28**	**+3.7V**
14	AXE	**29**	**PWR_GND**
15	GPIO1	**30**	**+3.7V**

$\overline{\text{ON}}$ pin is used to start up the GSM module by connecting it to ground for at least 1 second and then release. By checking the START line, we can know whether the GSM module is powered on or not. If START line is high after 900 ms from $\overline{\text{ON}}$ pin release, it means that the module is powered on. This two pin is connected in to the PIC for turn-on the GSM module.

The first step to test GM862 module is to send the command AT. If we get the answer OK, then we are fine and can start to send other AT-command. As it has autobauding embedded, we just fix our baud rate to send the data. The recommended speed is 38400, 8, N, 1 (38400 baud, 8 bits without parity bit, logic inverted and 1 stop bit). However, with the PIC microcontroller, we cannot reach this high speed with low error, so we only use 9600, 8, N, 1 or 19200, 8, N, 1 (See TABLE III).

Among the AT-command set, some are interesting for our project. They are:

- **AT**: Make the device set the right speed and character format of the serial port.

  Send: AT

  Receive: AT
  OK

- **AT+CSCA?** : Report the current value of the default Service Center Address (SCA), then store in the RAM and use it while sending SMS.

  ```
  AT+CSCA?

  AT+CSCA?  +CSCA:  "6616110400",
  145
  ```

  From the modem's answer, the SMSC number is 6616110400. Other information is not interesting.

- **AT+CMGF=0**: Select the SMS format to be used in reading and writing message. For this setting, PDU mode is selected because most of mobile phone, which has build in modem, supports this mode. For text mode, it is difficult to find such mobile phone. For example, ERICSSON T68 support only PDU mode. We can check it with **AT+CMGF=?** as show below

```
AT+CMGF=?

AT+CMGF=?
+CMGF: (0)
OK
```

Normally, if the modem supports both PDU and Text mode, the modem should reply like: `+CMGF: (0,1)` and then we can select which mode we want by using the command as below (in here, PDU mode is selected):

```
AT+CMGF=0

AT+CMGF=0
OK
```

- **AT+CPMS="ME"**: Select memory storages to be used for reading, writing, sending and storing received SMS. For this setting, Mobile Equipment internal storage (ME) is used. However, this setting can be used only with mobile phone ERICSSON T68, it cannot work with GM862 because the memory for received SMS storing is "SM" (SIM SMS memory storage). So, if we use ERICSSON T86 as wireless link module, **AT+CPMS="ME"** is used, otherwise **AT+CPMS="SM"** is used. For other mobile phone, please contact its AT-command set for more detail. To check the available setting on the mobile phone, **AT+CPMS=?** is used as shown below.

```
AT+CPMS=?

AT+CPMS=?
+CPMS: ("ME","SM"),("ME","SM"
),("ME")
OK
```

From this reply, we can know that the memory from which message are read and delete can be in the mobile equipment message storage (ME) or in the SIM message storage card (SM), the memory to which writing and sending operations are made can be in ME or SIM but the memory to which received SMs are preferably to be stored is in ME only, therefore the following command is used to select the memory location.

```
AT+CPMS="ME"
```

```
AT+CPMS="ME"
+CPMS:  25,70,0,30,25,70
OK
```

- **AT+CMGL=0**: List all the new incoming message stored into <memr>, which is **ME** for ERICSSON T68 and **SM** for GM862. If no new message is detected in the memory, the modem will reply only OK.

```
AT+CMGL=0
```

```
AT+CMGL=0  OK
```

In case there is a new message in the memory, the modem will indicate the memory location (in here is 35), status of the message (0 if new, 1 if already read) and the number of byte in the TPDU is 39 follow by the message and OK. For the message, we will analyze it in section 3.2.5.

```
AT+CMGL=0
```

```
AT+CMGL=0  +CMGL:  35,0,,39
069166611301400040A91664949872
6000050208261902282184369929A
1C0699A0A4135AA58288456A71482
D1243
OK
```

- **AT+CMGS=<length>:** Send to the network a SMS message input as a PDU. The parameter <length> can be a number from 8 to 176, which represents the length of the PDU to be sent in bytes. For more information about PDU format, see pack78/unpack78 algorithm in section 3.2.5.

```
AT+CMGS=34
0691666111400001000A916649498
726000018
4369929A1C0699A0A4135AA582884
56A71482D124300
```

After sending this command to the modem, the same data echoes from the modem, without following by any message, to indicate that SMS has been send successfully, otherwise an error message is followed as below

```
AT+CMGS=34
0691666111400001000A916649498
726000018
4369929A1C0699A0A4135AA582884
56A71482D124300

+CMS ERROR: 500    (in case of error)
```

- **ATD <PhoneNumber>:** Dial a given phone number in Data Call mode (Circuit Switch Data Transfer). We use this command to send the data from mobile node to the central computer at baud rate 9600 bits per second.
- **ATA**: Answer an incoming call (Voice or Data Call)
- +++**ATH:** Exit the data mode and enter the command mode then hang up the data call. For more information about ATD, ATA and +++ATH, see Exchange of Command and Response Signaling algorithm in section 3.2.4 and section 3.2.6.

Note that some AT-commands need to cooperate with other AT-commands in order to achieve a complete process such as ATD should be use with ATA and +++ATH to form a complete Data Call process, hence these 3 AT-commands will be descript later at Exchange of Command and Response Signaling algorithm. Our Modify HyperTerminal uses Keystroke font. It is chosen because of its ability to differentiate one from another character even though it is not a printable character set such as character 26 (Ctr+Z) is displayed as a box. This information is very useful as we work with the string manipulation because it gives use the information of where to start/stop cutting a portion of string (take out only the useful information) from the long input string.

Some people may ask why don't we use AT+CNMI=x,x,x,x,x to indicate the Terminal Equipment that a new message is detected? And then use **AT+CMGR=0** to read all the new incoming SMS? The answer is that during Data Call process, a new message maybe received but at that time, the modem is busy with Data Call ⇒ that SMS is missing and maybe not only one SMS that is missing. With the command AT+CMGL=0, we can check the memory after we finish Data Call process, result no even one SMS is missed. In our application, SMS play a very important role in the hold system; it carries out the Alarm information, Report and Script command.

3.1.7 GPS Module

In this project, GPS Received Engine Board, from Starts Navigation Tech. Ltd, is used. This GPS module has many features such as:

- SiRF GPS Architecture (www.sirf.com)
- SiRF startII high performance and low power consumption chip set
- Support standard NMEA 0183 protocol
- All-in-view 12-channel parallel processing
- Snap Lock 100ms re-acquisition time
- Cold start under 45 seconds, average
- Superior urban canyon performance
- Foliage Lock for week signal tracking
- Optional build-in Super Cap to reserve system data for rapid satellite acquisition.
- Full-duplex RS-232 port for navigation and control messages
- Differential GPS capability through 2nd RS-232 port

For the electrical characteristics, we focus only on:

- Power:
    - Voltage supply : 3.8Vdc ~ 6.5Vdc
    - Current supply : 60mA typical for Continuous mode and 20 mA typical for Trickle power mode.

  - Backup Power : +2.5V to 3.6V
  - Backup Current: 10 μA typical
- Serial Port
  - Ports : one for GPS, one for DGPS
  - Electrical level : TTL level (ET-102), output voltage level : 0 ~3.5V RS-232 level (ER-102)
  - Communication: Full duplex asynchronous
  - Code type : ASCII
  - GPS protocol : SiRF binary/NMEA 0183 changeable (Default : NMEA)
  - GPS Output Message :
    - SiRF binary >> position, velocity, altitude, status and control
    - NMEA 0183 >> GGA, GSA, GSV, RMC (VTG and GLL are optional)
  - GPS transfer rate : Software command setting (Default : 4800bps for NMEA)
  - DGPS protocol : RTCM SC-104, ver 2.00, type 1, 2, and 9
- Dynamic Condition
  - Altitude : 18000 meters (60000 feet) max
  - Velocity : 515 meters/sec (1000 knots) max
- Accuracy
  - Position Horizontal
    - 15m 2d RMS (SA off)
    - 10m 2d RMS, WAAS enable (SA off)
    - 1 ~ 5 m, DGPS corrected
  - Velocity : 0.1m/sec 95% (SA off)
  - Time : 1 μs synchronized to GPS time

From these specifications, we can supply this module with 5Vdc and 3V back-up battery. As we also use one battery back-up for RTC, this battery can be use for both RTC and GPS module. For NMEA protocol, we only use GGA (Global Positioning System Fixed Data format) because this output command can provide us information about Latitude, N/S indicator, Longitude, E/W indicator and MSL Altitude in meter. Other information is not taking into account.

To control/polled out the output of standard NMEA message GGA, GLL, GSA, GSV, RMC and VTG, Query/Rate Control command is used. The format of this command is:

```
$PSRF103,<msg>,<mode>,<rate>,<cksumEnalbe>*CKSUM<CR><LF>
```

<msg>	0=GGA, 1=GGL, 2=GSA, 3=GSV, 4=RMC, 5=VTG
<mode>	0=SetRate, 1=Query
<rate>	Output every <rate> seconds, off=0, max=255
<cksumEnable>	0=disable Checksum, 1=Enable checksum for specified message

For our application, the below command is used.

```
$PSRF103,00,01,00,01*25
```

It means: Query the GGA message with checksum enable. By using this command message, standard NMEA message may be polled once. Periodic mode is not preferable for our application since everything is controlled by CPU module and we want this CPU go to low power mode and wake up at a predefine time (we have schedule and mask for the CPU to work). After sending this command to the GPS module, the GPS will reply with the message look like below string and its description/data format is shown on TABLE XI.

```
$GPGGA,161229.487,3723.2475,N,12158.3416,W,1,0,7,1.0,9.0,M,,,,0000*18
```

However, before using this control message to poll out the GGA's data, we need to disable all other data for they can not disturb us while reading and also to let GPS relax. As each command needs the checksum field, it makes us a little bit difficult to change form one to other control message. Facing with this problem, a Serial Communication program for

GPS is implemented. It generates automatically checksum field and adds <CR><LF> to the end of the string. Some serial communication programs do not generate <CR> or <CR><LF> to the terminal equipment. From our program, the above command messages are generated as below:

- Disable GGA (Global positioning system fixed data) message:

  $PSRF103,00,00,00,01*24

- Disable GLL (Geographic position - latitude / longitude) message:

  $PSRF103,01,00,00,01*25

- Disable GSA (GNSS DOP and active satellites) message :

  $PSRF103,02,00,00,01*26

- Disable GSV (GNSS satellites in view) message :

  $PSRF103,03,00,00,01*27

- Disable RMC (Recommended minimum specific GNSS data) message:

  $PSRF103,04,00,00,01*20

- Disable VTG (Course over ground and ground speed) message:

  $PSRF103,05,00,00,01*21

NMEA means National Maritime Electronic Association. The NMEA 0183 standard for interfacing marine electronic devices is a voluntary industry standard, first released in March of 1983. The NMEA 0183 standard defines electrical signal requirements, data transmission protocol, timing and specific sentence formats for a 4800 baud, 8-bit, no parity, one stop bit (8N1) serial data bus. The NMEA 0183 sentences are all printable ASCII character.

The data is transmitted in the form of "sentences". Each sentence starts with a "$", a two letter "talker ID", a three letter "sentence ID", followed by a number of data fields separated by commas, and terminated by an optional checksum, and a carriage return/line feed. A sentence may contain up to 82 characters including the "$" and CR/LF.

If data for a field is not available, the field is simply omitted, but the commas that would delimit it are still sent, with no space between them. Since some fields are variable width, or may be omitted as above, the receiver should locate desired data fields by counting commas, rather than by character position within the sentence.

The optional checksum field consists of a "*" and two hex digits representing the exclusive OR of all characters between, but not including, the "$" and "*". A checksum is required on some sentences. The standard allows individual manufacturers to define proprietary sentence formats. These sentences start with "$P", then a 3 letter manufacturer ID, followed by whatever data the manufacturer wishes, following the general format of the standard sentences.

Some common talker IDs are:

- GP : Global Positioning System receiver
- LC : Loran-C receiver
- OM : Omega Navigation receiver
- II : Integrated Instrumentation

Fig 35 shows the picture of the GPS module the dimension of the module. Table XII shows the Pin-out of the 20-pin interface connector of the GPS module.

TABLE XI
GGA DATA FORMAT

Name	Example	Units	Description
Message ID	$GPGGA		GGA protocol header
UTC Time	161229.487		hhmmss.sss
Latitude	3723.2475		ddmm.mmmm
N/S Indicator	N		N=north or S=south
Longitude	12158.3416		dddmm.mmmm
E/W Indicator	W		E=east or W=west
Position Fix Indicator	1		See Table B-3
Satellites Used	07		Range 0 to 12
HDOP	1.0		Horizontal Dilution of Precision
MSL Altitude[1]	9.0	meters	
Units	M	meters	
Geoid Separation[1]		meters	
Units	M	meters	
Age of Diff. Corr.		second	Null fields when DGPS is not used
Diff. Ref. Station ID	0000		
Checksum	*18		
<CR><LF>			End of message termination

1. SiRF Technology Inc. does not support geoid corrections. Values are WGS84 ellipsoid heights.

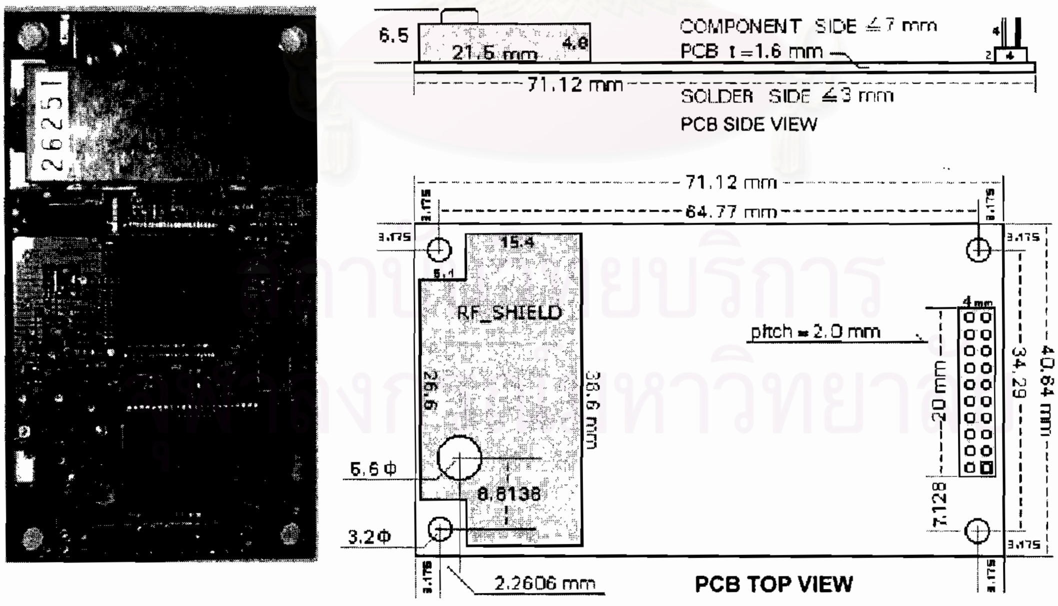


Fig 35. Dimension of GPS module

TABLE XII
PIN-OUT OF THE 20-PIN INTERFACE CONNECTOR

Pin Number	Name	Description	Type
1	VANT	Antenna DC Voltage	Input
2	VDC	3.8V~6.5V DC Power Input	Input
3	VBAT	Backup Battery	Input
4	VDC	(Shorted with pin 2)	Input
5	PBRES	Push Button Reset Input (Active Low)	Input
6	RESERVED	(Reserved)	
7	SELECT	Down-load data from RS232 to flash ROM (Reserved)	
8	RESERVED	(Reserved)	
9	RESERVED	(Reserved)	
10	GND	Ground	
11	TXA	Serial Data Output A (GPS Data)	Output
12	RXA	Serial Data Input A (Command)	Input
13	GND	Ground	
14	TXB	Serial Data Output B (No Used)	Output
15	RXB	Serial Data Input B (DGPS Data)	Input
16	GND	Ground	
17	RESERVED	(Reserved)	
18	GND	Ground	
19	TIMEMARK	1PPS Time Mark Output	Output
20	RESERVED	(Reserved)	

The battery backup is used to power the SRAM and RTC when main power is removed. Typical current draw is 10 μA. Without an external backup battery or Gold-capacitor, the module/engine board will execute a cold start after every turn on, resulting a long delay while turn on. To achieve the faster startup offered by a hot or warm start, either a battery backup mush be connected or a Gold-capacitor should be installed. To maximize the battery lifetime, the battery voltage should not excess the supply voltage and should be between 2.5V and 3.6V. All these specification are taken into account by our design.

3.1.8 Radio Link Module Hardware Design

As in Fig 11 shows the block diagram of the whole system and its functionalities are described in chapter 2, we can now start analyzing the schematic of our Radio Link module (not shown), however, algorithms are discussed in section 3.2. From this design, we can see that:

- The main power supply (7Vdc ~ 10Vdc) is regulated by a regulator L7805. This regulator contains many protection features such as current limitation, thermal, etc, which makes this regulator become hard to be destroyed. The polarization protection is warranted by a diode and noise reduction is warranted by other 3 condensers. However the non-polarized condenser should be connected as near as possible to the PIC to reduce the noise efficiently as it eliminates the high frequency noise.
- Power-on Reset is improved by using the NPN transistor C1815 and other auxiliary component such as condenser and resistor. Once press the RESET button, the CPU module and Radio link module and GPS module will be reset. We can see that this reset system is separated one from other. The reason is that, our system is embedded with ICSP. To facilitate the design, each module has its own ICSP hence imply two reset systems. In the CPU design, the reset system of GPS is connected with CPU module, without any separation like this case.
- Have ICSP embedded to facilitate the programmer. With ICPS, we can change the executable code any time we want to update it, that implies the flexibility and mass production support.
- Have LCD for display the inside operations and other information for the user.

- Embedded with debug capability. The connector J-DEBUG provides the programmer enough connection to debug/monitor the internal process, program the EEPROM and PIC. For debug pin, a resistor 1 kΩ is used to limit the current in case of short circuit at PC side. In here, the RS-232 level is inverted in software; no need hardware level converters. For I^2C device programming, two resistor 4.7 kΩ are used to poll up SDA and SCL pin to high
- The connector J-CPU is used to communicate with CPU module. This connector provides enough connection to communicate with the CPU and other I^2C extension module. With this design, we can update/change the Radio Link or CPU module without interfere each other function. This implies the flexibility to update/adapt to the new demand/application. The communication between Radio Link and CPU is supported by 5 pins (I2C, Flag, RxTx, SDA and SCL); other pins are used for ICSP, debug, reset and LED indicator.
- The internal processes are, not only displayed via LCD but also indicated by LEDs. LED-Red is used to indicate the PIC's process with mobile phone such as reading/sending SMS, check new incoming SMS and check incoming call whereas LED-Green is used to indicate the status of I^2C bus used in Radio Link module. LED-Yellow indicates the CPU operation. It will turn on every second but with a different turn-on delay. This delay depends on the CPU process, if CPU is turns to work, the LED is turn on, otherwise it is turned off for low-power mode. Turn On/Off operations of GSM module are controlled in software. As this GSM module is designed to work with PC (inverted TTL level), a level converter is needed because we use hardware serial port embedded in the PIC that cannot be change the level by software anymore. In here, MAX232 is used.

3.1.9 CPU Module Hardware Design

The CPU module, for this design:

- The power supply is taken from Radio Link module via J-RADIO connector. As describe above, this connector allows the CPU module to communicate with Radio Link, ICSP, I^2C programmer, PC and LED indicator. A resistor 1 kΩ is used to limit the current in case of short circuit at PC side.

- The analog sensors can be connected to PIC via J-ANAIN whereas the digital switches can be connected to PIC via J-DIGIN.
- The output board (analog and digital) can be connected to PIC via J-DIGANA. As described in chapter 2, we provide 8 digital output pins and 2 analog output pins and it is another board, not included in this schematic. To communicate with that output board, J-OUTBOARD is used to connect all the digital & analog output pins and I^2C bus (SDA, SCL) from CPU board to Output board. We do like this because we want all (input pins and output pins) on the CPU board so we can see directly that the input/output pins are controlled by CPU module but the most important reason is to be easy to update the Output board without changing the output connector (modular system).
- As our process is synchronized with the time, a RTC is used to generate 1 Hz square-wave. Normally, this 1 Hz signal can be taken from GPS module. However, input/output module and GPS module are optional and can be changed easily, and we want CPU board work independently, not depends on GPS module $\Rightarrow$ RTC is needed. This RTC is not only used for generating 1 Hz clock but also for synchronizing the storing data with real time calendar. This information is also used to display the date and time on the LCD for user to see/control whether the module is working or not working. Note that, initially, the RTC needs to be set up for it can know what time it is now and which frequency it should generate, so an indicator for this RTC is need as it plays a very important role in the CPU process. LED-Blue will turn on at the output frequency of the RTC. To keep these data on the RTC, a backup battery is needed. In here, a small 3V Lithium battery is used.
- One EEPROM (24C512) is used to store the data (configuration data and input data from sensor).

3.1.10 Input Board

This input board is made for testing purpose. It is used for measuring the temperature and luminosity. In this design (not shown), two types of temperature sensor IC's are used. The first one is LM335 whereas the second is LM35.

The LM335 is a precision temperature sensor [18], easily calibrated, integrated circuit temperature sensor. Operating as 2-terminal zener, the LM335 has a breakdown voltage directly proportional to absolute temperature at +10 mV/°K. With less then 1Ω dynamic impedance, the device operates over a current range of 400 μA to 5 mA with virtually no change in performance. When calibrated a 25°C the LM335 has typically less then 1°C error over 100°C temperature range. Unlike other sensors the LM335 has a linear output, which makes interfacing to readout or control circuitry especially easy. The LM335 operates from –40°C to +100°C. For more information about this low cost temperature sensor, visit National Semiconductor at www.national.com.

The LM35 series are precision integrated-circuit temperature sensors [18], whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Centigrade scaling. The LM35 does not require any external calibration or trimming to provide typical accuracies of ±1/4°C at room temperature and ±3/4°C over a full -55 to +150°C temperature range. Low cost is assured by trimming and calibration at the wafer level. The LM35's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. It can be used with single power supplies, or with plus and minus supplies. As it draws only 60 μA from its supply, it has very low self-heating, less than 0.1°C in still air. The LM35 is rated to operate over a -55° to +150°C temperature range, while the LM35C is rated for a -40° to +110°C range (-10° with improved accuracy). For more information about this low cost temperature sensor, visit National Semiconductor at www.national.com.

For luminosity sensor (photo resistor), we have no datasheet or information about its characteristic so we just measures its value in dark place and under sunlight to see the resistance variation. From this measure, a complete testing board is made as shown below.

This design is to adapt to the analog input characteristic described in section 3.1.1 (maximum input impedance is 10 kΩ). The other condensers and resistor in this design are used a filter purpose (low pass filter) to eliminate the noises (electromagnetic noise, high frequency noise from the power supply and Poisson noise from the light source for photo resistor). For LM35, a series R-C damper from output to ground is used to improve the tolerance of capacitance of the heavy capacitive load (LM35 can drive only 50 pf capacitive load). As this is a simple design for testing the system performance, the result is not so good but enough to see the evaluation of the surrounding environment.

From our experiments, we can say that LM335 is difficult to use comparing with LM35 because LM335 needs calibration, which is difficult to adjust by hand; moreover, if we want it to have small error (less then 1% over 100°C range), we have to calibrate it at 25°C. However, LM35 is about 3 times more expensive then LM335.

3.1.11 Output Board

As describe in chapter 2, Two PCF8591 are used to assure the two analog outputs with 8-bit resolution and one PCF8574A is used to assure the 8 digital outputs. The power switching board with 8 outputs can be connected with this 8 digital output directly.

The design of digital output is easy, however, for analog output, we need a very good amplifier that can operate with 5V power supply, low power, high input impedance, low output impedance and rail-to-rail input/output feature such as LMC6001 or MCP6001 [19].

The MCP6001 device, from Microchip, has a high phase margin, which makes it ideal for capacitive load applications. The low supply voltage, low quiescent current and wide bandwidth make the MCP6001 ideal for battery-powered applications.

However, we cannot find this kind of IC in Thailand and as it does not interfere our system performance testing, we do not develop this output board for instance.

3.1.12 RS-232 to I^2C Debug & Programming Board

This board (not shown) is design to help developer easily debug the Radio Link module and CPU module. This board is not only used for debug but also program the EEPROM and RTC at first use. The EEPROM needs the configuration data for work with CPU and Radio

Link module whereas RTC need the initial time calendar to run its real time calendar. This board is also used for programming the PIC.

The ICSP programmer board is not part of RS232-I2C board but it is related to our development work, so we just show it, not explain. For the software (IC-Prog), we can download from http://kudelsko.fr/prog_pic/sommaire.htm for free of charge.

The PIC serial programmer board can program EEPROM as well as PIC16F8X, PIC16X62X, PIC16X55X, PIC16C6X and PIC1687X but we still need RS-232 to I^2C interface adaptor because the PIC serial programmer cannot program RTC and we need to change the cable from serial port (use with Mobile phone) to parallel port (use with EEPROM) ⇔ not convenient for developer and user.

3.2 Communication Protocol & Algorithm

3.2.1 I^2C-Bus

In consumer electronic, telecommunications and industrial electronics, there are often many similarities between seemingly unrelated designs. For example, nearly every system includes:

- Some intelligent control, usually a single-chip microcontroller
- General-purpose circuits like LCD drivers, remote I/O ports, RAM, EEPROM, or data converters
- Application-oriented circuits such as digital turning and signal processing circuits for radio and video systems, or DTMF generators for telephones with tone dialing.

To exploit these similarities to the benefit of both systems designers and equipment manufacturers, as well as to maximize hardware efficiency and circuit simplicity, Philips developed a simple bi-directional 2-wire bus for efficient inter-IC control. This bus is called the Inter IC or I^2C-bus. At present, Philips's IC range includes more then 150 CMOS and bipolar I^2C-bus compatible types for performing functions in all tree categories (Version 1.0 - 1992, Version 2.0 - 1998, Version 2.1 - 2000). All I^2C-bus compatible devices incorporate an on-chip interface, which allows them to communicate directly with each other via I^2C-bus. This design concept solves the many interfacing problems encountered when designing digital control circuits. Here are some of the features of the I^2C-bus:

- Only two bus lines are required; a serial data line (SDA) and a serial clock line (SCL)
- Each device connected to the bus is software addressable by a unique address and simple master/slave relationships exist at all times; masters can operate as master-transmitters or as master-receivers
- It's a true multi-master bus including collision detection and arbitration to prevent data corruption if two or more masters simultaneously initiate data transfer
- Serial, 8-bit oriented, bi-directional data transfers can be made at up to 100 kbit/s in the Standard-more, up to 400 kbit/s in the Fast-mode, or up to 3.4 Mbit/s in the High-speed mode
- On-chip filtering rejects spikes on the bus data line to preserve data integrity
- The number of ICs that can be connected to the same bus is limited only by a maximum bus capacitance of 400 pF.

Designer benefits: I^2C-bus compatible ICs allow a system design to rapidly progress directly from a functional block diagram to a prototype. Moreover, since they 'clip' directly onto the I^2C-bus without any additional external interfacing, they allow a prototype system to be modified or upgraded simply by 'clipping' or 'unclipping' ICs to or from the bus. Here are some of the features of I^2C-bus compatible ICs, which are particularly attractive to designers:

- Functional blocks on the block diagram correspond with the actual ICs; designs proceed rapidly from block diagram to final schematic
- No need to design bus interfaces because the I^2C-bus interface is already integrated on-chip
- Integrated addressing and data-transfer protocol allow systems to be completely software-defined
- The same IC types can often be used in many different applications

- Design-time reduces as designers quickly become familiar with the frequently used functional blocks represented by I^2C-bus compatible ICs
- ICs can be added to or removed from a system without affecting any other circuits on the bus
- Fault diagnosis and debugging are simple; malfunctions can be immediately traced
- Software development time can be reduced by assembling a library of reusable software modules.

In additional to these advantages, the CMOS ICs in the I^2C-bus compatible range offer designers some special features, which are particularly attractive for portable equipment and battery-backed systems. They all have:

- Extremely low current consumption
- High noise immunity
- Wide supply voltage range
- Wide operating temperature range

The I^2C-bus concept: The I^2C-bus supports any IC fabrication process (NMOS, CMOS, bipolar). Two wires, SDA and SCL, carry information between the devices connected to the bus. Each device is recognized by a unique address (whether it is a microcontroller, LCD driver, memory or keyboard interface) and can operate as either a transmitter or receiver, depending on the function of the device. Obviously an LCD driver is only a receiver, whereas a memory can both and receiver and transmit data. In additional to transmitters and receivers, devices can also be considered as masters and receivers, devices can also be considered as masters or slaves when performing data transfers (see Table XIII). A master is the device, which initiates a data transfer on the bus and generates the clock signal to permit that transfer. At that time, any device addressed is considered a slave.

The I^2C-bus is a multi-master bus. This means that more then one device capable of controlling the bus can be connected to it. As masters are usually micro-controllers, let's consider the case of a data transfer between two microcontrollers connected to the I^2C-bus (see Fig 36).

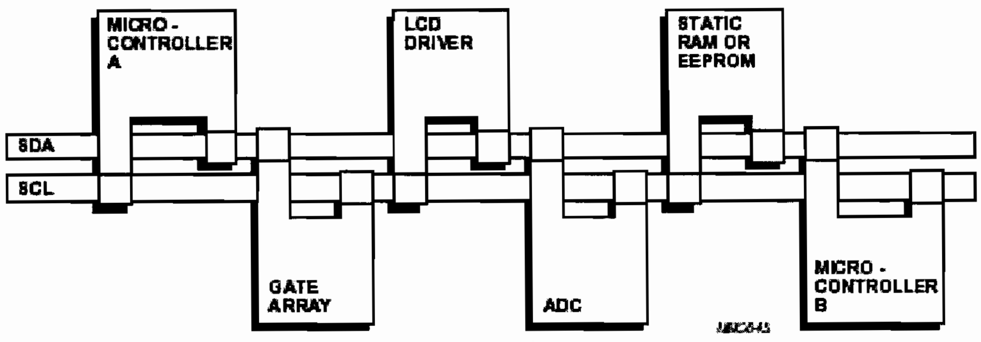


Fig 36. I^2C-bus Configuration Using Two Microcontrollers

This highlights the master-slave and receiver-transmitter relationships to be found on the I^2C-bus. It should be noted that these relationships are not permanent, but only depend on the direction of data transfer at that time. The transfer of data would proceed as follows:

- Suppose microcontroller A wants to send information to microcontroller B:
    - Microcontroller A (master), addresses microcontroller B (slave)
    - Microcontroller A (master-transmitter), sends data to microcontroller B (slave-receiver)
    - Microcontroller A terminates the transfer
- If microcontroller A wants to receive information from microcontroller B:
    - Microcontroller A (master) addresses microcontroller B (slave)
    - Microcontroller A (master-receiver) receives data from microcontroller B (slave-transmitter)
    - Microcontroller A terminates the transfer.

Even in this case, the master (microcontroller A) generates the timing and terminates the transfer. Only master can generate the clock signal on the I^2C-bus whatever it is master-transmitter or master-receiver.

The possibility of connecting more then one microcontroller to the I^2C-bus means that more then one master could try to initiate a data

TABLE XIII
DEFINITION OF I^2C-BUS TERMINOLOGY

TERM	DESCRIPTION
Transmitter	The device which send data to the bus
Receiver	The device which receives data from the bus
Master	The device which initiates a transfer, generates clock signals and terminates a transfer
Slave	The device addressed by a master
Multi-master	More then one master can attempt to control the bus at the same time without corrupting the message
Arbitration	Procedure to ensure that, if more than one master simultaneously tries to control the bus, only one is allowed to do so and the winning message is not corrupted
Synchronization	Procedure to synchronize the clock signals of two or more devices

transfer at the same time. To avoid the chaos that might ensue from such an event, an arbitration procedure has been developed. This procedure relies on the wired-AND connection of all I^2C interfaces to the I^2C-bus.

If two or more masters try to put information onto the bus, the first to produce a 'one' when the other produces a 'zero' will lose the arbitration. The clock signals during arbitrations are a synchronized combination of the clocks generated by the masters using the wired-AND connection to the SCL line. For more detailed information concerning arbitration, see I^2C-bus specification from Philip because it is out of our scope. Even though in this project, we have two masters but we do not use this arbitration to control the I^2C-bus. We implement a modify I^2C-bus for our application as descript in the next section.

General characteristics: Both SDA and SCL are bi-directional lines, connected to a positive supply voltage via a current-source or pull-up resistor (see Fig 37). When the bus is free, both lines are HIGH. The output stages of devices connected to the bus mush have an open-drain or open collector to perform the wired-AND function.

Bit transfer: Due to the variety of different technology devices (CMOS, NMOS, bipolar), which can be connected to the I^2C-bus, the levels of the logical '0' (LOW) and '1' (HIGH) are not fixed and depend on the associated level of V_{DD}. One clock pulse is generated for each data bit transferred.

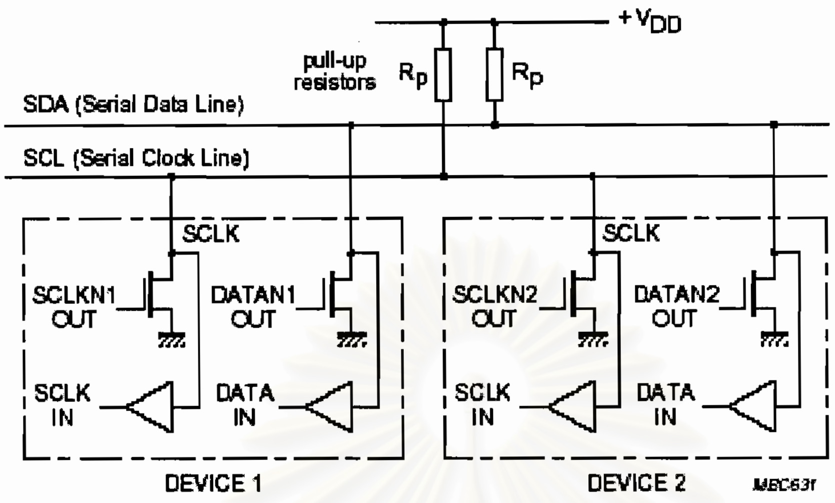


Fig 37. Connection of Standard and Fast-mode Devices to the I^2C-bus

Data validity: The data on the SDA line must be stable during the HIGH period of the clock. The HIGH or LOW state of the data line can only change when the clock signal on the SCL line is LOW (see Fig 38)

START and STOP conditions: Within the procedure of the I^2C-bus, unique situations arise which are defined as START (S) and STOP (P) condition (see Fig 38).

- A HIGH to LOW transition on the SDA line while SCL is HIGH is one such unique case. This situation indicates a START condition.
- A LOW to HIGH transition on the SDA line while SCL is HIGH defines as STOP condition.

START and STOP conditions are always generated by the master. The bus is considered to be busy after the START condition. The bus is considered to be free again a certain time after the STOP condition. The bus stays busy if a repeated START (Sr) is generated instead of a STOP condition.

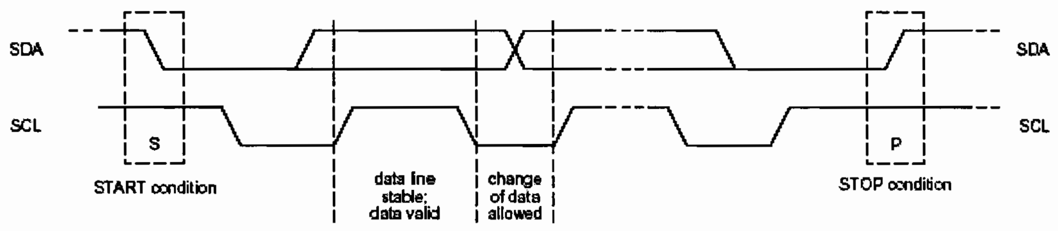


Fig 38. Start Condition, Bit Transfer and Stop Condition for I^{2}C-bus

Transferring data: every byte put on the SDA must be 8-bits long. The number of bytes that can be transmitted per transfer is unrestricted. Each byte has to be followed by an acknowledge bit. Data is transferred with the most significant bit (MSB) first (Fig 39). If a slave can't receive or transmit another complete byte of data until it has performed some other function, for example servicing an internal interrupt, it can hold the clock line SCL low to force the master into a wait state. Data transfer then continues when the slave is ready for another byte of data and releases clock line SCL.

In some cases, it's permitted to use a different format from the I^{2}C-bus format (for CBUS compatible devices for example). A message which starts with such an address can be terminated by generation of a byte. In this case, no acknowledge is generated.

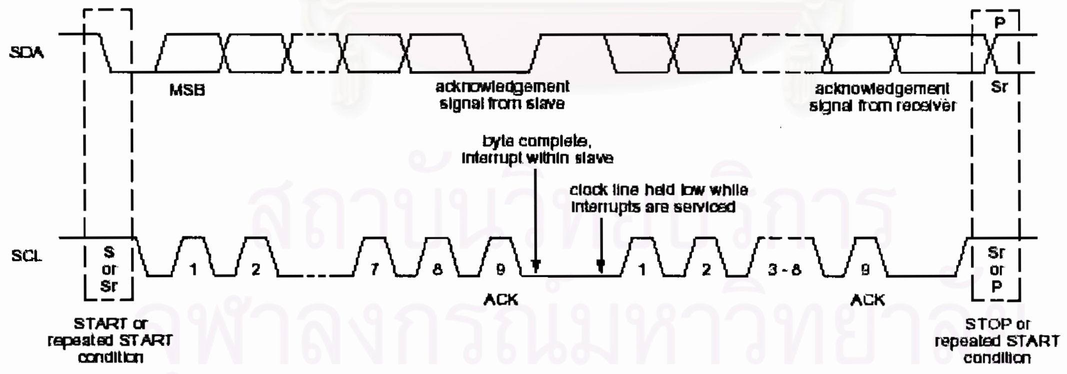

SDA
MSB
acknowledgement signal from slave
byte complete, interrupt within slave
clock line held low while interrupts are serviced
acknowledgement signal from receiver
P
Sr
SCL
S or Sr
1
2
7
8
9
ACK
1
2
3 - 8
9
ACK
Sr or P
START or repeated START condition
STOP or repeated START condition

Fig 39. Data Transfer on the I^{2}C-bus

Acknowledge: Data transfer with acknowledges is obligatory. The acknowledge-related clock pulse is generated by the master. The transmitter release the SDA line (HIGH) during the acknowledge clock pulse. The receiver mush pull down the SDA line during the acknowledge clock pulse so that it remains stable LOW during the HIGH period of this clock pulse (see Fig 40). Of course, set-up and hold times mush also be taken into account.

Usually, a receiver which has been addressed is obliged to generate an acknowledge after each byte has been received, except when the message start with a CBUS address. When a slave doesn't acknowledge the slave address (for example, it's unable to receive or transmit because it's performing some real-time function), the data line must be left HIGH by the slave. The master can then generate either a STOP condition to abort the transfer, or a repeated START condition to start a new transfer.

If a slave-receiver does acknowledge the slave address but, some time later in the transfer cannot receive any more data bytes, the master mush again abort the transfer. This is indicated by the slave generating the not-acknowledge on the first byte to follow. The slave leaves the data line HIGH and the master generates a STOP or repeated START condition.

If a master-receiver is involved in a transfer, it mush signal the end of data to the slave-transmitter by not generating an acknowledge on the last byte that was clocked out of the slave. The slave-transmitter must release the data line to allow the master to generate a STOP or repeated START condition.

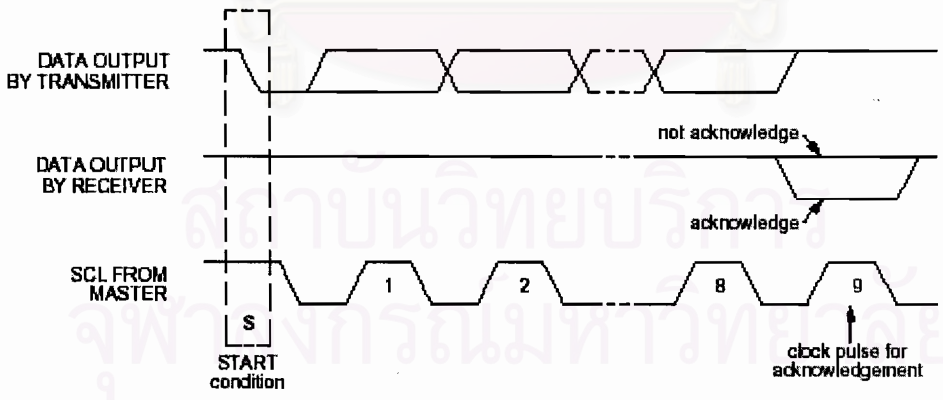


Fig 40. Acknowledge on the I^2C-bus

3.2.2 Modify I^2C-Bus

Looking back to Fig 11, we can see that our system is I^2C-bus multi-master mode; however, this mode is not used in this system to exchange the information between Radio Link and CPU module. We create another protocol to do this function.

First of all, let us explain why the I^2C-bus multi-master mode is not used in this system? The reason is that:

- At first, our microcontroller are not supported I^2C hardware interface, so this multi-master mode become a very difficult task to do in software since we have sample at less twice time the clock frequency to detect the falling edge or rising edge of the SDA to detect the start and stop condition. If each microcontroller do only this task, it is not difficult to implement this algorithm, but since each microcontroller have to monitor the input (for CPU) and especially the incoming call and message (Radio Link), this task becomes harder to implement. Moreover, if the incoming call and message are synchronized with the CPU process (sampling the input data), it is not difficult to implement this algorithm as well, but since this process is taken randomly with different time delay, this task becomes very hard to implement. To solve this problem, our algorithm is implemented.
- After we find PIC16F8XX, the multi-master mode is solved. However, during sending/receiving data process, if CPU detects some errors and sends that information via I^2C-bus to Radio Link, what will happen? The hardware interrupt does not occur because during sending/reading process (especially update data process because we have to wait for the update data sending by central node and these data are very important because it characterizes the mobile node process), all the interrupt sources are disable, so the information sent by CPU will be loss if there are more then two byte are sent because the hardware register can handle only 2 bytes (SSPBUF and SSPSR). So, as we've already implemented the modify I^2C protocol and as it works well, we continue to use this protocol for our system.

We still use I^2C-bus to send out the exchange information but with additional 3 lines and one byte in the EEPROM at address 7h.

- **I2C** line: is used to indicate both microcontrollers that the I^2C-bus is busy. We loss one line just for keeping this information but we can write a very simple program to check the I^2C-bus whether it is busy or not, no need to sample the I^2C-bus at twice time clock frequency. The main problem is not this twice time clock frequency and how to know that the I^2C-bus is free, but how to solve the problem when these two microcontrollers become both master-transmitter and try to address the slave device (EEPROM and RTC or other I^2C slave device), which one will get token to uses I^2C-bus? Right?

- **Flag** line: is used by microcontroller A to read the acknowledge signal from other microcontroller B that it has some thing to tell microcontroller A and already noted on the white board (EEPROM, location 7h). After reading and executing the command on the white board, microcontroller A have to clean the white board (clear the correspond bit on the EEPROM (7h)).
- **RxTx** line: is used by microcontroller B to tell microcontroller A that it has some thing to tell and already noted on the white board (EEPROM, locate at 7h). This line is set to LOW as acknowledge signal is generated and will bet set to HIGH when microcontroller B check there is nothing on the white board.

The RxTx line of microcontroller A is connected to Flag line of microcontroller B and vise versa.

The design of these Flag line and RxTx line is came from SDA line (bi-directional), however, we would like to separate this SDA line into two as Flag line (for receiving the signal) and RxTx line (for transmitting the signal). The same reason, we loss two lines but we can use a very simple algorithm to control the exchange information between these two microcontrollers.

Each information is written as binary format on the white board, if it is '1', the information is true otherwise the information is invalid (see Fig 41).

For more detail information about the process of each microcontroller with the white board, see section 3.2.7 and 3.2.8.

So, to exchange the information from one microcontroller to other, I^2C-bus protocol is used with additional 3 pins and one byte of EEPROM. With this design, each microcontroller can run the task asynchronously and come back to read/write the data on the I^2C device like multi-master mode with a simple algorithm to implement into the PIC.

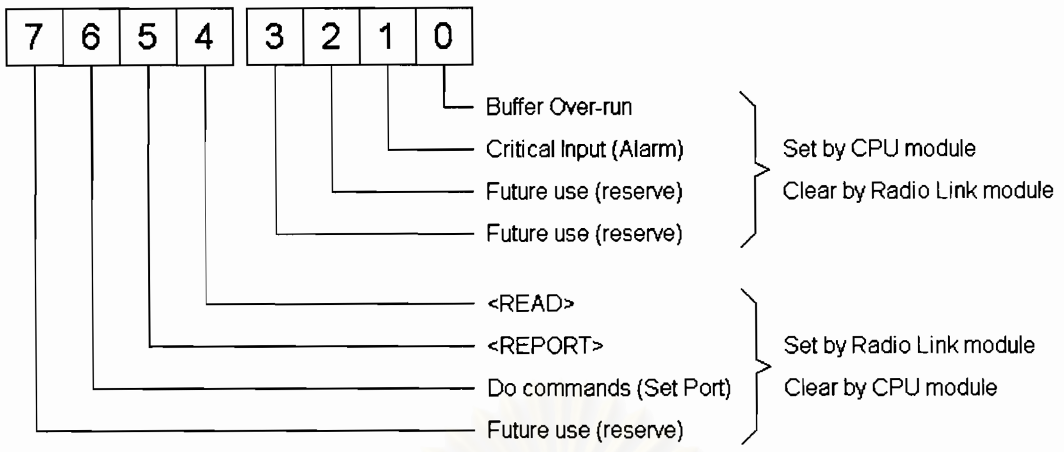


Fig 41. Format of EEPROM (7h)

3.2.3 RS-232

RS–232 is a "complete" standard. This means that the standard sets out to ensure compatibility between the host and peripheral systems by specifying:

1) Common voltage and signal levels
2) Common pin wiring configurations
3) A minimal amount of control information between the host and peripheral systems.

Unlike many standards which simply specify the electrical characteristics of a given interface, RS–232 specifies electrical, functional, and mechanical characteristics in order to meet the above three criteria. However, we just take some of these aspects of the RS–232 standard to be discussed below.

Electrical Characteristics: The original RS–232 standard was defined in 1962. As this was before the days of TTL logic, it should not be surprising that the standard does not use 5 volt and ground logic levels. Instead, a high level for the driver output is defined as being +5 to +15 volts and a low level for the driver output is defined as being between –5 and –15 volts. The receiver logic levels were defined to provide a 2 volt noise margin. As such, a high level for the receiver is defined as +3 to

+15 volts and a low level is –3 to –15 volts. Fig 42 illustrates the logic levels and logic waveform defined by the RS–232 standard. It is necessary to note that, for RS–232 communication, a low level (–3 to –15 volts) is defined as a logic 1 and is historically referred to as "marking". Likewise a high level (+3 to +15 volts) is defined as logic 0 and is referred to as "spacing".

The RS–232 standard also limits the maximum slew rate at the driver output. This limitation was included to help reduce the likelihood of cross–talk between adjacent signals. The slower the rise and fall time, the smaller the chance of cross talk. With this in mind, the maximum slew rate allowed is 30 V/ms. Additionally, a maximum data rate of 20k bits/second has been defined by the standard. Again with the purpose of reducing the chance of cross talk, the impedance of the interface between the driver and receiver has also been defined. The load seen by the driver is specified to be 3kW to 7kW.

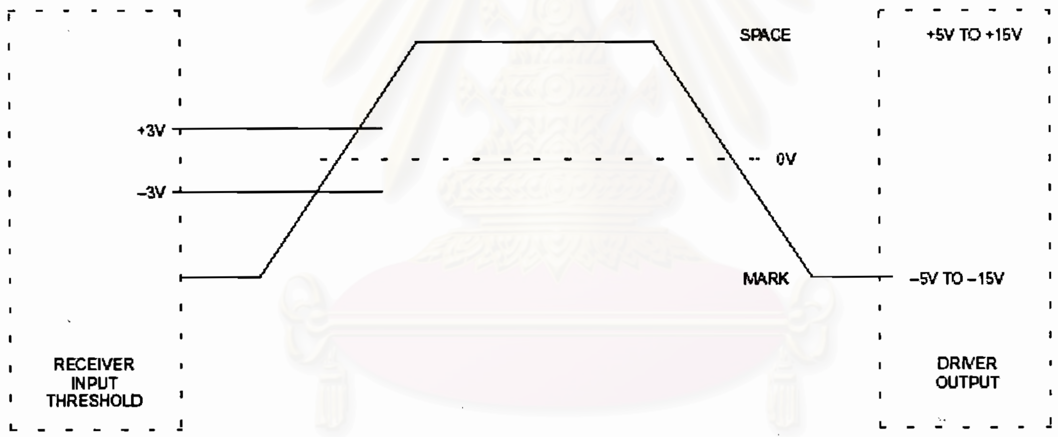


Fig 42-A. RS-232 Logic Levels

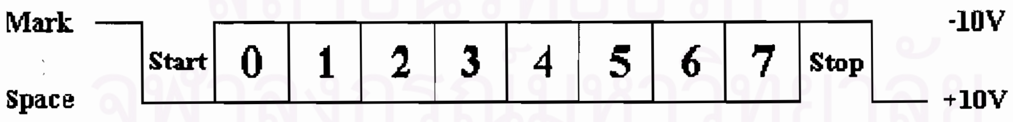


Fig 42-B. RS-232 Logic Waveform

For the original RS–232 standard, the cable between the driver and the receiver was also specified to be a maximum of 15 meters in length. This part of the standard was changed in revision "D" (EIA/TIA–232–D). Instead of specifying the maximum length of cable, a maximum capacitive load of 2500 pF was specified which is clearly a more adequate specification. The maximum cable length is determined by the capacitance per unit length of the cable, which is provided in the cable specifications.

Fig 43 shows the RS-232 waveforms in TTL or CMOS level. RS-232 communication is asynchronous. That is a clock signal is not sent with the data. Each word is synchronized using its start bit, and an internal clock on each side, keeps tabs on the timing.

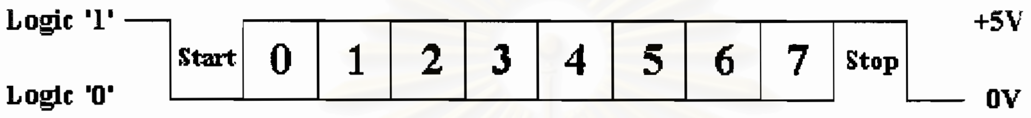


Fig 43. TTL/CMOS Serial Logic Waveform

The diagram above shows the expected waveform from the UART when using the common 8N1 format. 8N1 signifies 8 Data bits, No Parity and 1 Stop Bit. The RS-232 line, when idle is in the Mark State (Logic 1).

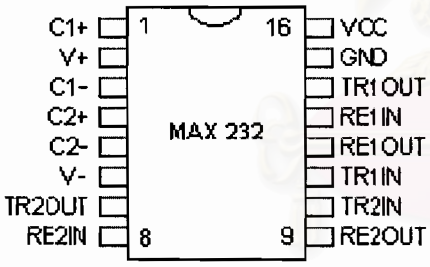
A transmission starts with a start bit, which is (Logic 0). Then each bit is sent down the line, one at a time. The LSB (Least Significant Bit) is sent first. A Stop Bit (Logic 1) is then appended to the signal to make up the transmission. The Fig 43 shows the next bit after the Stop Bit to be Logic 0. This must mean another word is following, and this is its Start Bit. If there is no more data coming then the receive line will stay in its idle state (logic 1).

We have encountered something called a "Break" Signal. This is when the data line is held in a Logic 0 state for a time long enough to send an entire word. Therefore, if you don't put the line back into an idle state, then the receiving end will interpret this as a break signal. The data sent using this method, is said to be *framed*. That is the data is *framed* between a Start and Stop Bit. Should the Stop Bit be received as Logic 0, then a framing error will occur. This is common, when both sides are communicating at different speeds.

Almost all digital devices, which we use, require either TTL or CMOS logic levels. Therefore the first step to connecting a device to the RS-232 port is to transform the RS-232 levels back into 0 and 5 Volts. As we have already covered, this is done by RS-232 Level Converters.

Two common RS-232 Level Converters are the 1488 RS-232 Driver and the 1489 RS-232 Receiver. Each package contains 4 inverters of the one type, either Drivers or Receivers. The driver requires two supply rails, +7.5 to +15v and -7.5 to -15v. As you could imagine this may pose a problem in many instances where only a single supply of +5V is present. However the advantages of these IC's are they are cheap.

Another device is the MAX-232. It includes a Charge Pump, which generates +10V and -10V from a single 5v supply. This IC also includes two receivers and two transmitters in the same package. This is handy in many cases when you only want to use the Transmit and Receive data Lines. You don't need to use two chips, one for the receive line and one for the transmit line. However all this convenience comes at a price, but compared with the price of designing a new power supply it is very cheap.



Pinouts for the MAX-232, RS-232 Driver/Receiver.

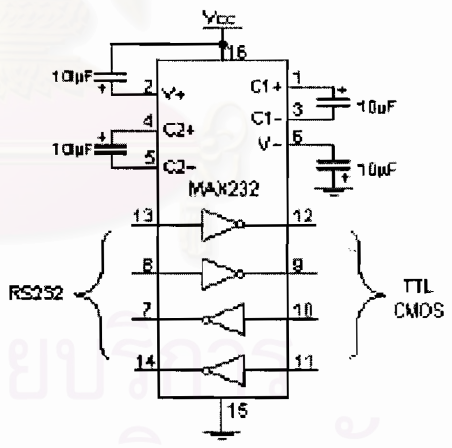


Fig 44. Typical MAX-232 Circuit

There are also many variations of these devices. The large values of capacitors are not only bulky, but also expensive. Therefore other devices are available which use smaller capacitors and even some with inbuilt capacitors. *(Note: Some MAX-232 can use 1 μF Capacitors).* However, the MAX-232 is the most common, and thus we will use this RS-232 Level Converter in our system.

Functional Characteristics: RS–232 has defined the function of the different signals that are used in the interface. These signals are divided into four different categories: common, data, control, and timing. TABLE XIV illustrates the signals that are defined by the RS–232 standard. As can be seen from the TABLE there is an overwhelming number of signals defined by the standard. The standard provides an abundance of control signals and supports a primary and secondary communications channel. Fortunately few applications, if any, require all of these defined signals. For example, only eight signals are used for a typical modem. Some simple applications may require only four signals (two for data and two for handshaking) while others may require only data signals with no handshaking. The complete list of defined signals is included here as a reference, but it is beyond the scope of this document to review the functionality of all of these signals.

Mechanical Interface Characteristics: In particular, RS–232 specifies a 25–pin connector. This is the minimum connector size that can accommodate all of the signals defined in the functional portion of the

TABLE XIV
RS-232 DEFINED SIGNALS

CIRCUIT MNEMONIC	CIRCUIT NAME*	CIRCUIT DIRECTION	CIRCUIT TYPE
AB	Signal Common	–	Common
BA	Transmitted Data (TD)	To DCE	Data
BB	Received Data (RD)	From DCE	
CA	Request to Send (RTS)	To DCE	Control
CB	Clear to Send (CTS)	From DCE	
CC	DCE Ready (DSR)	From DCE	
CD	DTE Ready (DTR)	To DCE	
CE	Ring Indicator (RI)	From DCE	
CF	Received Line Signal Detector** (DCD)	From DCE	
CG	Signal Quality Detector	From DCE	
CH	Data Signal Rate Detector from DTE	To DCE	
CI	Data Signal Rate Detector from DCE	From DCE	
CJ	Ready for Receiving	To DCE	
RL	Remote Loopback	To DCE	
LL	Local Loopback	To DCE	
TM	Test Mode	From DCE	
DA	Transmitter Signal Element Timing from DTE	To DCE	Timing
DB	Transmitter Signal Element Timing from DCE	From DCE	
DD	Receiver Signal Element Timing From DCE	From DCE	
SBA	Secondary Transmitted Data	To DCE	Data
SBB	Secondary Received Data	From DCE	
SCA	Secondary Request to Send	To DCE	Control
SCB	Secondary Clear to Send	From DCE	
SCF	Secondary Received Line Signal Detector	From DCE	

*Signals with abbreviations in parentheses are the eight most commonly used signals.
**This signal is more commonly referred to as Data Carrier Detect (DCD).

standard. The pin assignment for this connector is shown in Fig 45. The connector for DCE equipment is male for the connector housing and female for the connection pins. Likewise, the DTE connector is a female housing with male connection pins. Although RS–232 specifies a 25–position connector, it should be noted that often this connector is not used. This is due to the fact that most applications do not require all of the defined signals and therefore a 25–pin connector is larger than necessary. This being the case, it is very common for other connector types to be used. Perhaps the most popular is the 9–position DB9S connector, which is also illustrated in Fig 45. This connector provides the means to transmit and receive the necessary signals for modem applications, for example. For our application, only TD (Tx), RD (Rx) and Ground (GND) line are used between the Air-modem (mobile phone or GSM module) and PC/Radio Link Module.

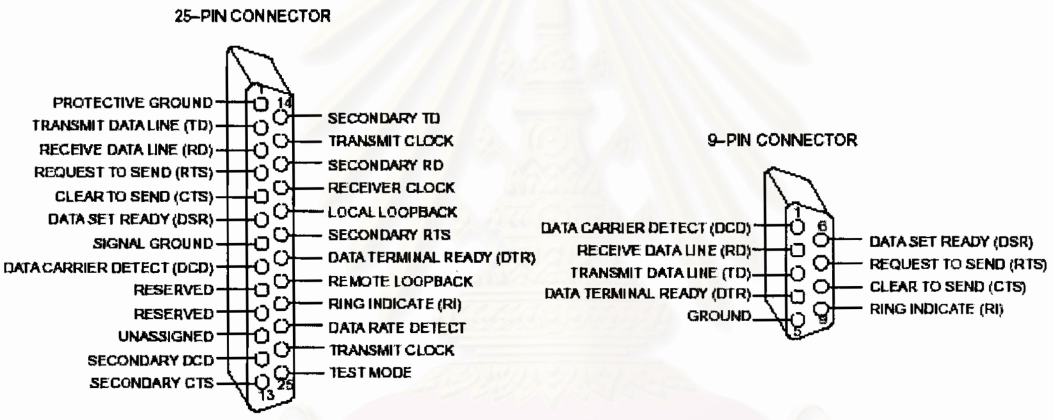


Fig 45. RS-232 Connector Pin Assignments

3.2.4 PC to Mobile Phone and Mobile Phone to Radio Link Module Exchange Protocol

At this exchange protocol level, AT-Command plays a very important role as it is used to establish the connection and disconnect the communication. Section 3.1.6 already mention about this command but not in detail. Now let see how we use this command to establish/disconnect the connection.

To establish a connection, PC (DTE) sends the command **ATD <phonenumber><CR>** to the mobile phone (DCE) that is connected to PC via serial port, where **phonenumber** is phone number to be dialed.

After send this command, the PC should wait for response from modem. The possible responses are:

- CONNECT 9600 : means that the call modem is now on line and the exchange data can be started
- BUSY : means that the line called is busy and the communication can not be established. If the PC wants to connect to the mobile node, the PC needs to retry again later.
- NO ANSWER : means that the receiver did not answer the call and the communication can not be established. So try again later.
- NO CARRIER : means that the modem handshaking has not been successful, so the user needs to check for mobile registration and signal strength and eventually retry.

So, it is necessary for the PC (central node) to check whether the response **CONNECT 9600** has been received or not, if not, the connection does not establish.

Suppose now that this step can be done successfully, and then we go forward to see what happen at mobile node. When an incoming call is detected, the modem (DCE) at mobile node will report an unsolicited code to the Radio Link module (DTE), which may be:

- RING : means the extended format of incoming call indication is disabled and a call (voice or data) is incoming
- +CRING: VOICE : means the extended format of incoming call indication is enabled and a voice call is incoming
- +CRING: ASYNC : means the extended format of incoming call indication is enabled and an asynchronous transparent data call is incoming
- +CRING: REL ASYNC : means the extended format of incoming call indication is enabled and an asynchronous reliable (not transparent) data call is incoming

• +CRING: SYNC	: means the extended format of incoming call indication is enabled and a synchronous reliable (not transparent) data call is incoming
• +CRING: FAX	: means the extended format of incoming call indication is enabled and a fax call is incoming.

By detecting a **RING** code, the Radio Link can now answer to this call by sending the command **ATA<CR>** and then wait for response:

• CONNECT 9600	: means the incoming call was a DATA one and called modem is now on line and then the exchange data can now start.
• ERROR	: means no incoming call is found, call may have been lost.
• NO CARRIER	: means the incoming call was a DATA one and the modem handshaking has not been successful so we need to check for mobile registration and signal strength and modem settings.
• OK	: means the incoming call was a VOICE call and is now active and then the communication can now start.

Note that after the PC calls the MN, even the MN can receive a call with **RING** code, but it does not mean that the communication is established even though after the MN answers with **ATA<CR>** command. After the MN answers to the call, both sides have to wait for a response **CONNECT 9600** send by the provider (see Fig 46).

After both sides detect **CONNECT 9600** response, the exchange data can now start. Chapter 2 already mentioned about this exchange data process. Now suppose that everything is finish and the mobile node want to disconnect the connection, +++ATH<CR> is used to exit the data mode and enter the command mode then hang up the data call.

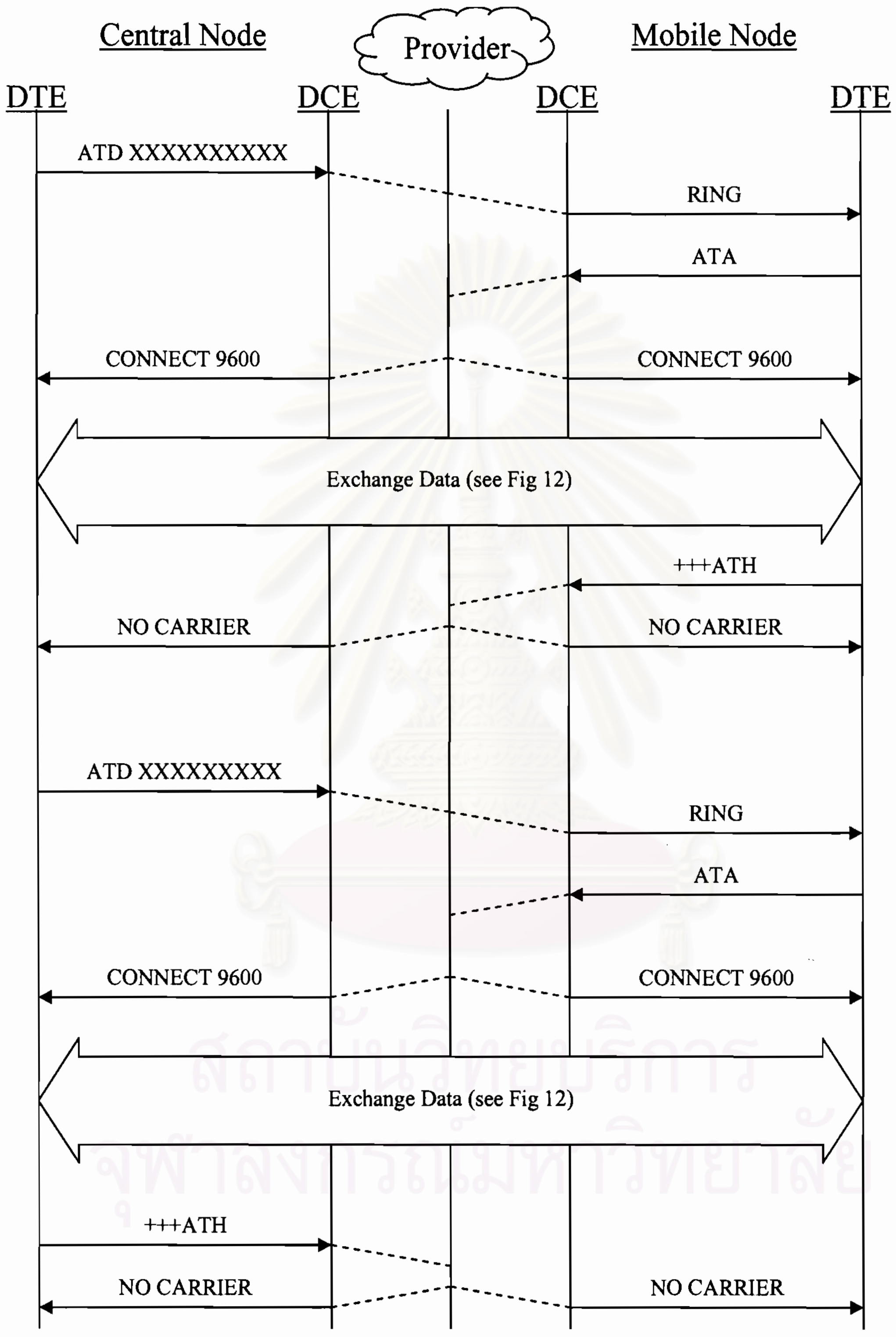


Fig 46. DTE-DCE Exchange Protocol

We can note that only the central node can ask for a connection to be established and only the mobile node decides whether to disconnect or not because only the mobile node knows when to disconnect (finish sending all the data out from EEPROM). However, to prevent an unpleasant case, the central node has right to disconnect the connection if there is no data received in a period predefined by us (typically 2s).

This predefined delay is chosen with the reason that some time, the modem at central node receives data and also updates the data into the buffer with practically unlimited size, but the program itself don't know whether the data has been updated or not because of the multitasking supported by the PC (the PC may not only run our program but also other program and so if the PC is not fast enough, this problem is occurred). To prevent this 2s monitoring time is chosen. This 2s monitoring time is used to monitor whether the incoming data has been changed or not, if yes, the timer will be reset otherwise the timer will start until it reach 2s value, and then the connection will be disabled by the central node.

3.2.5 SMS

SMS is the abbreviation for Short Message Service. SMS is a way of sending short message to mobile telephones and receiving short messages from mobile telephones. "Short" means a maximum of 160 bytes. According to the GSM Association, "Each short message is up to 160 characters in length when Latin alphabets are used and 70 characters in length when non-Latin alphabets such as Arabic and Chinese are used".

The message can consist of text character, in which case the message can be read and written by human beings. SMS text messages have become a staple of wireless communications in Europe and Asia/Pacific and are gradually gaining popularity in North America. The message also can consist of sequences of arbitrary 8-bit bytes, in which case the message probably is created by a computer on one end and intended to be handled by a computer program on the other.

The part of our application on the computer/mobile node creates an SMS message to be sent to the mobile node/computer. This message is handed off to the short message center of our local telephone company which telephone number of the mobile we want it to send to. The telephone company finds the mobile and passed the SMS message to it.

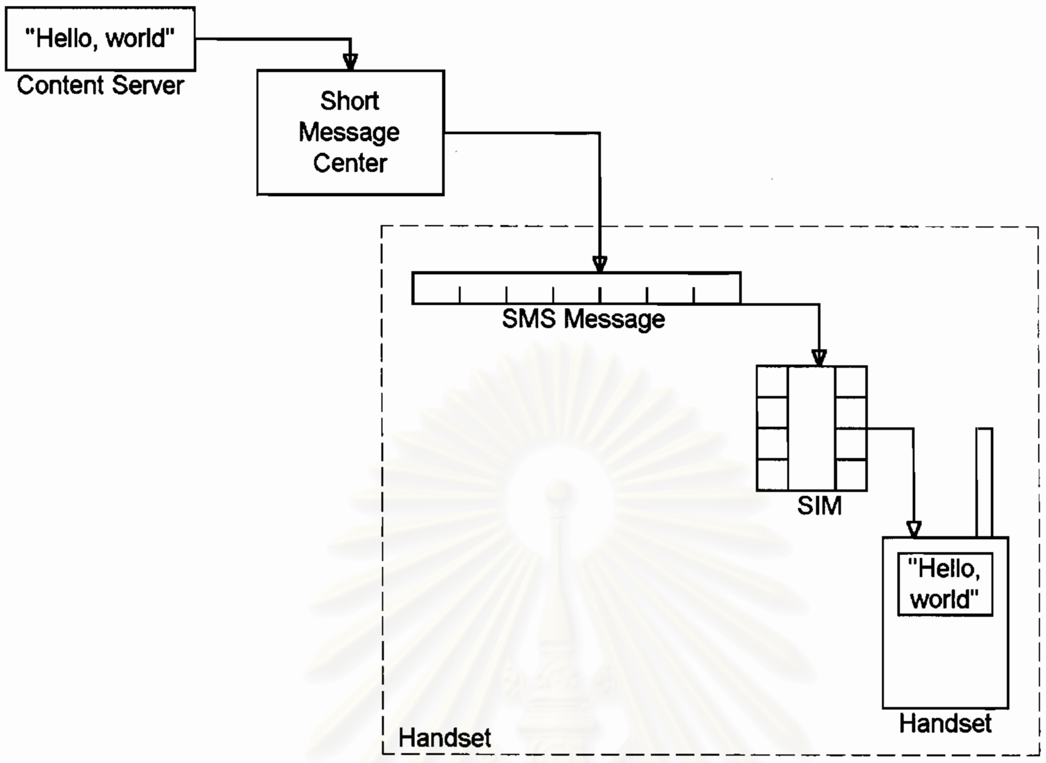


Fig 47. Message Flow from Server to Screen

The message has a flag set in it that tells the handset to pass the message to the SIM (Subscriber Identity Module). The message also has a flag that says which application on the SIM should receive the message. When the SIM received the message from the handset, it checks to see which application to give it to and hands it off to the mobile side of our application. Fig 47 illustrates the flow of traffic.

Receiving a message works exactly the same way, only in reverse. The mobile side of our application generates an SMS message, attaches the telephone number of our air modem and hands it over to the handset. The handset passes it to the network that delivers it to our PC/mobile node. See also Fig 48.

Because the mobile network is an active participant in moving messages between our application and a mobile device, we have to be much more concerned with the details of formatting the messages we send. Remember the mobile network actually looks at the bytes in our message (actually in the headers on our message) to figure out what to do with it. We will discover that there are lots of things besides who should

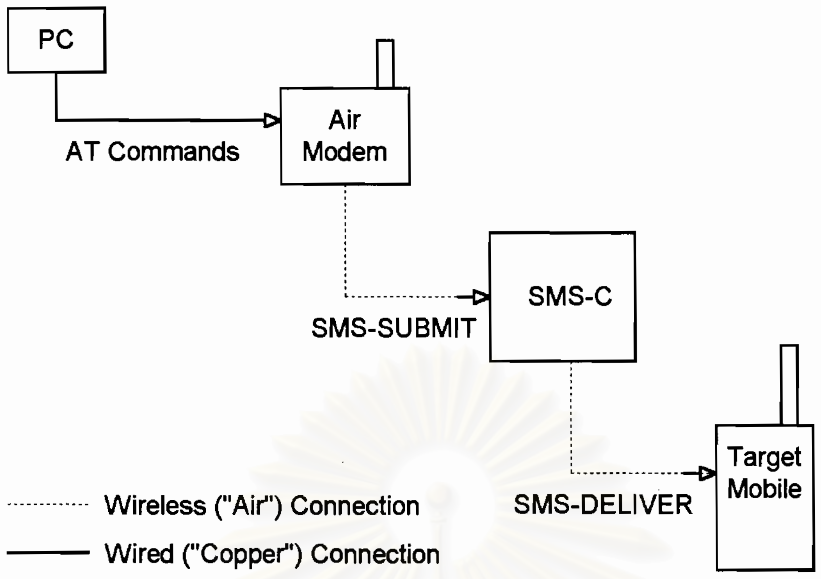


Fig 48. Message Flow from PC to Mobile Handset

receive the message that we can tell the GSM network and its SMS contain not only the message but also lots of other information that instructs the networks as to how and when we want this to happen.

The two standards that govern the construction of SMSs what we will be using are:

- 3GPP 23.040 : Technical realization of SMS
- 3GPP 24.011 : PP SMS support on the mobile radio interface

These standards cover the encoding of the message that gets delivered to the destination handset and the encoding of the instructions to the GSM network and the SMSC. Fig 49 shows the complete SMS header diagram. We build all the headers, so we will have to remember whom we are talking to and what we are saying to them as we build our SMS message.

Let's start by sending a simple "Hello, world" message to the mobile phone at **+66 50 48 32 51**. What we do is pack in a hex-encoded byte blob all the information needed to get this message to its destination along with the message itself and ship this blob off to the carrier's SMSC which in turn will get it to where it is going.

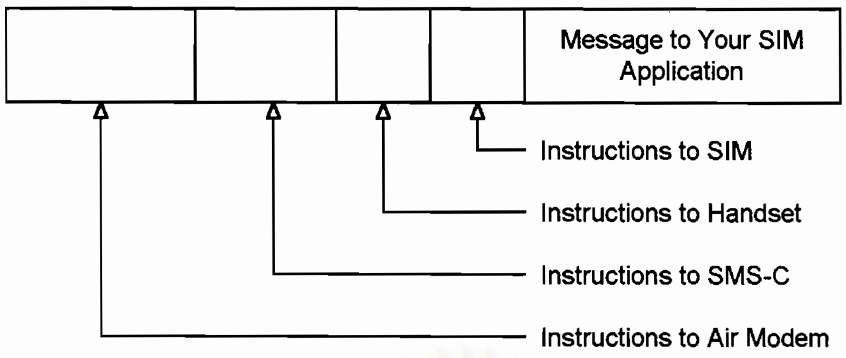


Fig 49. SMS Message Headers

The byte blob is an SMS_SUBMIT Transfer Protocol Data Unit (TPDU) that consists of the following fields:

1) Transfer protocol parameters = 0x01 (an SMS_SUBMIT TPDU)
2) Message reference number = 0x00 (let the handset assign it)
3) Length of destination number in digits = 0x0A (10 digits)
4) Type of destination number = 0x91 (international format)
5) Destination telephone number (nibble swapped) = 0x6605842315
6) Protocol identifier = 0x00 (implicit)
7) Data coding scheme = 0x00 (GSM default alphabet)
8) Message length = 0x0C (there are 12 character in "Hello, world")
9) Message = 0xC8329BFD6681EE6F399B0C ("Hello, world")

The coding of the actual message, "Hello, world", requires some explanation. No stone is left unturned when it comes to optimizing the use of the air interface. If we had transmitted the ASCII characters as bytes, we would has wasted a bit for every character we sent because ASCII character are coded on 7 bits and sending this message as an 8-bit byte wastes 1 bits. Now, 1 bit is not big deal if we have megabytes of memory and gigabytes of disk space, but on an air interface this represents a waste of one-eighth of the channel capacity and this cannot be tolerated.

What we do is very simple (see Fig 50). First, we put the first character into the first byte. Next, we take the low-order bit of the seven bits of the second ASCII character and stuff it in the unused high-order bit of the first byte. Now we put the six remaining bits of the second character into the second byte. Next, we take the two low-order bits of the seven bits of the third ASCII character and stick them into the two unused high-order bits in the second byte, and so forth. Here is the result of applying this packing algorithm to "Hello, world":

Unpacked	H	e	l	l	o
Unpacked	1001000	1100101	1101100	1101100	1101111
Packed	11001000	00110010	10011011	11111101	01100110
Packed	C8	32	9B	FD	66

Unpacked	,		w	o	r
Unpacked	0101100	0100000	1110111	1101111	1110010
Packed	10000001	11101110	01101111	00111001	10011011
Packed	81	EE	6F	39	9B

Unpacked	l	d
Unpacked	1101100	1100100
Packed	00001100	
Packed	0C	

So the complete SMS_SUBMIT TPDU for "Hello, world" looks like this:

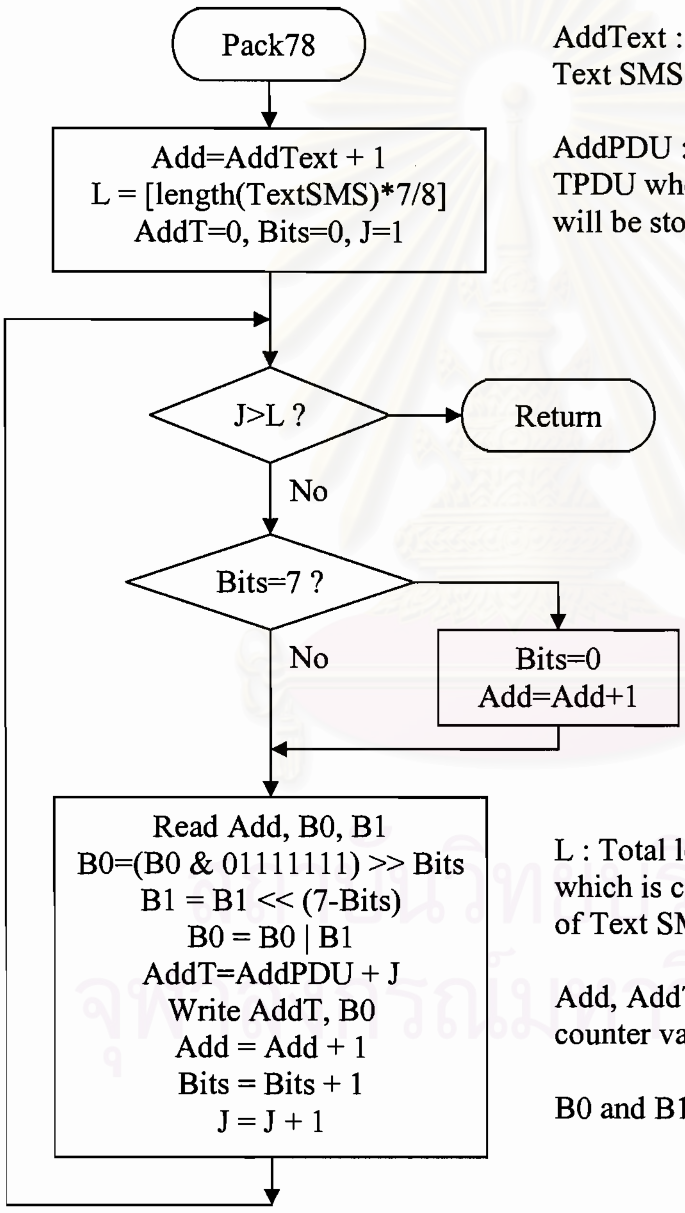
```
01000A9166058423150000CC8329BFD6681EE6F399B0C
```

All we have to do now is use an AT command to send this TPDU off to the SMSC. This is the send-message AT command:

```
AT+CMGS=<TPDU_length><CR>
<SMSC_address><TPDU><CTRL-Z>
```

The SMSC address is the telephone number of the SMSC to which the handset should send the TPDU. Like the destination telephone number, the telephone number of the SMSC consists of three sub-fields:

1) Length of the telephone number in octets = 0x06 (1+5 = 6 octets)
2) Format of the SMS telephone number = 0x91 (international format)
3) Telephone number of SMSC (nibble swapped) = 0x6661114000 (5 octets)



AddText : First address of the Text SMS where it is stored.

AddPDU : First address of the TPDU where the packed data will be stored.

L : Total length of the TPDU which is calculate from length of Text SMS (before Packing)

Add, AddT, Bits and J are counter variables.

B0 and B1 are variables

Fig 50. Packing Algorithm

This particular SMSC is in the VoiceStream network, where the handset set we are using as our air modem is registered. This phone number is belonging to DTAC Mobile Phone Company. If we subscribe to other provider, this number should be change to that SMSC number. TABLE XV shows the SMSC of some company that we found it from Internet.

TABLE XV
SMSC NUMBER ASSOCIATED WITH THE PROVIDER AND COUNTRY

Country	Provider	SMSC Number
Cambodia	MOBITEL	85512000024
	SAMART	85516800000
Belgium	MOBISTAR	3295955205
	MOBISTAR	32495002530
	PROXIMUS	32475161616
	ORANGE	32486000005
France	SFR	33609001390
	BOUYGUE	33660003000
	ITINERI	33689004000
	ITINERI	33689004431
	ITINERI	3368900458
Indonesia	TELKOMSEL	6281100000
	EXELCOMINDO	62818445009
Malaysia	ADAM	60173600010
	CELCOM	60193900000
	MUTIARA	60162999000
	MAXIS	60120000015
New Zealand	VODAFONE	6421600600
Pakistan	MOBILINK	92300000042
Singapore	MOBILE ONE	6596845999
	MOBILE ONE	6596845997
	SINGTEL	6596400001
	SINGTEL	6596500001
	SINGTEL	6598189999
	SINGTEL	6596197777
	STARHUB	6598540020
Thailand	AIS	6618310808
	DTAC	6616110400

Finally, the following code is what we write to the modem in order to send "Hello, world" to the mobile phone +66 50 483 251:

```
AT+CMGS=23<CR>
06916661114000
01000A91660584231500000CC8329BFD6681EE6F399B0CΞ
```

Ξ is ASCII character 26 (Uppercase Greek xi), which is equivalent to CTRL-Z

The corresponding unpacking algorithm that we need when receive message from the handset is shown in Fig 51.

When the mobile phone receives a message, we can use AT+CMGL=0 to list the entire unread message or just to get the index and then use AT+CMGR=index to read one by one. As you can see how AT+CMGL=0 works better comparing with AT+CMGR, that is why we decide to use this command at central node as well as at mobile node.

For decoding the incoming SMS, it is not so different. Let's see together for message mentioned in section 3.1.6:

- If AT+CMGL=0 was used:

```
AT+CMGL=0

AT+CMGL=0   +CMGL:   35,0,,39
069166611301400040A91664949872
60000502082619022821843699 29A
1C0699A0A4135AA582884 56A71482
D1243

OK
```

After received a string +CMGL:, the number 35 indicates the memory location, in where the incoming SMS was stored. This index is very useful for us because we will use it to delete the correspond message in the memory to prevent out of memory in the mobile phone. It is also used to read the message with the command AT+CMGR as shown in the next paragraph. The following number 0 indicate the status of the message. The number 0 means this is a received message that has not been read, and if it is number 1, it means this received message has been read already. The number 39 indicates the TPDU length in text mode.

- If AT+CMGR=35 was used (35 is the index of the SMS in the memory. We get it from AT+CMGL=0 as show above)

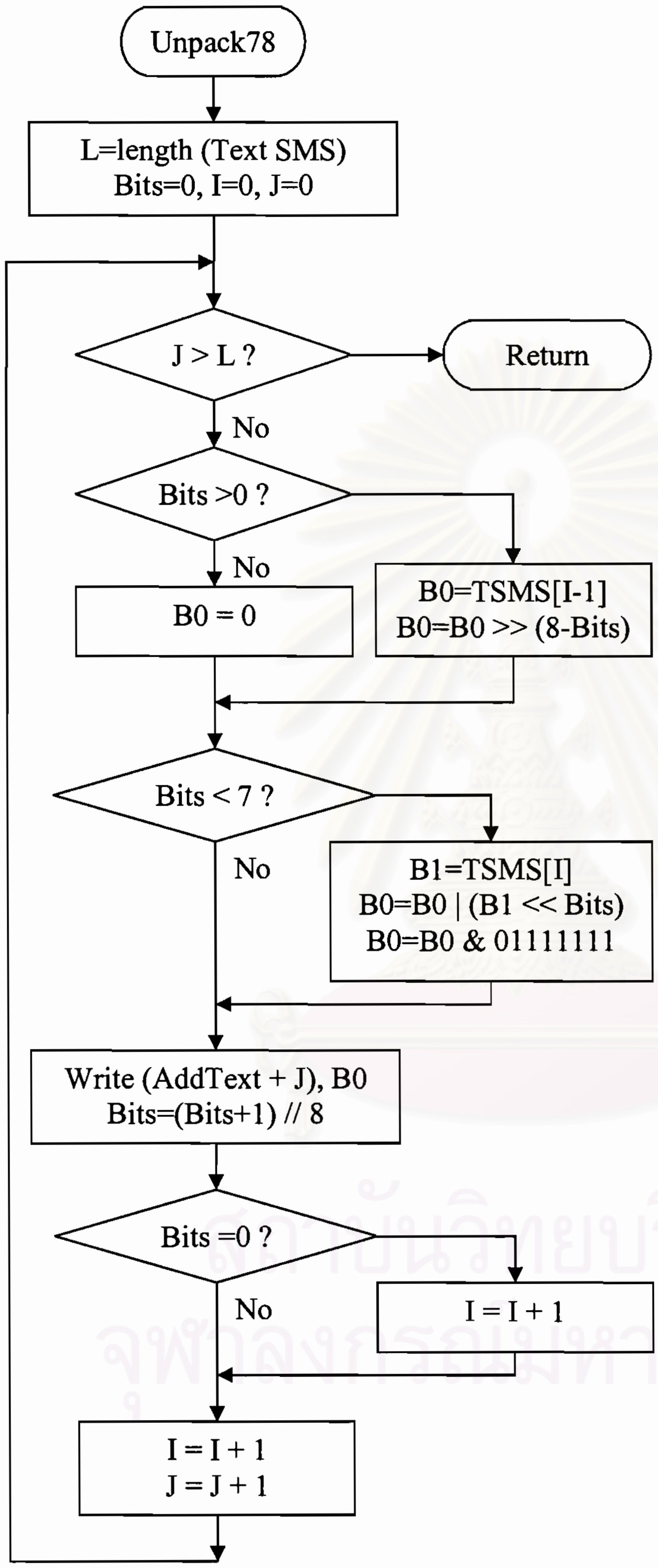
```
AT+CMGR=35

AT+CMGR=35   +CMGR:   1,,39
06916661130140040A91664949872
6000050208261902282184369929A
1C0699A0A4135AA58288456A71482
D1243

OK
```

After received the string "+CMGR:" the following number 1 indicates that this message has been read already. And for number 39, it indicates the same meaning as in AT+CMGL=0.

From these two AT commands, we can see that the SMS_DELIVER TPDU is the same. That is:

```
06916661130140040A91664949872
6000050208261902282184369929A
1C0699A0A4135AA58288456A71482
D1243
```

It is in the same general format as the data in the AT+CMGS command, namely the SMSC telephone number followed by a TPDU. In this case, however, it is the telephone number of the SMSC delivering the message and an SMS_DELIVER TPDU rather then and SMS_SUBMIT TPDU. In other words, the TPDU is being delivered to the handset rather then the handset submitting a TPDU to the network. We will discover that what is delivered is not exactly the same as what is submitted.



TSMS : is a buffer that store TPDU from serial port.

AddText : is the first address from where the unpacked strings are stored.

L : is the length of unpacked string. It is read directly from TPDU.

Bits, I and J are counter variables

B0 and B1 are variables

Fig 51. Unpacking Algorithm

The SMSC phone number is just like the one we send to, so let's analyze the SMS_DELIVER TPDU. We will be using 3GPP TS 23.040 standard to do this.

1) Transfer protocol parameters = 0x04 (SMS_DELIVER with no more coming)
2) Length of original address = 0x0A (10 bytes)
3) Type of original address = 0x91 (international format)
4) Original address (nibble swapped) = 0x6649498726 (+6694947862)
5) Protocol identifier = 0x00
6) Data coding scheme = 0x00
7) Service center timestamp (nibble swapped) = 0x50208261902282 (Y/M/D/H/M/S/Zone = 2005 February 28, 16:09:22, GMT-7)
8) Length of message = 0x18 (18h = 24 ASCII character = [24*7/8] = 21 bytes)
9) Message = 4369929A1C0699A0A4135AA58288456A71482D1243 ("CRITICAL INPUT DETECTED!")

Now we can verify that 39 = (1+1+1+5+1+1+7+1+21).

3.2.6 Radio Link Module

So far, we only talk about the components and standard protocols/algorithms plus some modifications. Now, let see our algorithms implementing in this module. As picture can descript more then letter, we will explain our algorithm by flowchart. These flowcharts are direct translation from the PICBasic Pro. One function/Procedure in PICBasic Pro will be represented by one flowchart. However, we will not take all function and procedure to show in this limited report, we take only the important one.

Main Program: This is the program that will run endlessly (see Fig 52). It starts by initial the GSM module and set the mobile settings, by using AT command, such as set baud rate, read Service Center Number and store in RAM for use while sending SMS, select SMS format and select the memory zone to store incoming SMS. See section 3.1.6 for the detail of AT command set.

After initialize the handset, we initialize the variables, display and some pins, for use in the next step. At this time, we also check the present of I^2C programmer by looking at I2C pin. If this connection is detected (I2C is HIGH), we change the SDA and SCL line to input/high impedance (z) in order to let I^2C-bus free.

The endless loop (main loop) is started from this point. We begin by checking LSB of DataF and then execute the corresponding task (send corresponding SMS) set by this variable (variable in EEPROM ($7), use for I^2C-bus multi-mater mode, see section 3.2.2). After checking and executing (CheckRxFlag), we clear the corresponding bit and update it back to the white board (EEPROM ($7)) for CPU to check and clear the acknowledge pin (set RxTx to HIGH). Next step is to read date and time from RTC and display it on LCD, and then check 4-MSB of DataF to see whether it is clear or not. If it is cleared, we will set our RxTx pin to HIGH, otherwise keep it LOW (CheckComFlag).

From this point, we start to work with modem. We have to wait about 4s to detect the code "RING", if after 4s is passed and no code is detected, we start to check the incoming SMS, otherwise we jump to Send-Update Data procedure. Checking SMS's process takes about 0.7s. If unread SMS is detected, we start to decode and delete this message as describe in section 3.2.5. After decoding this SMS and storing the message, in text format, in the corresponding EEPROM, we execute our script function to do what the script wants us to do. If <READ> and/or <REPORT> command is detected, we will set the corresponding bit in DataF and update back to the white board. At this time, we also check if the 4-MSB of DataF is clear or not, if it is cleared, we start to run SendSMS function (see Send SMS section for more detail) otherwise; we set our RxTx pin to LOW. Finally, we jump back to the main loop and do like this forever.

Read SMS: See section 3.2.5

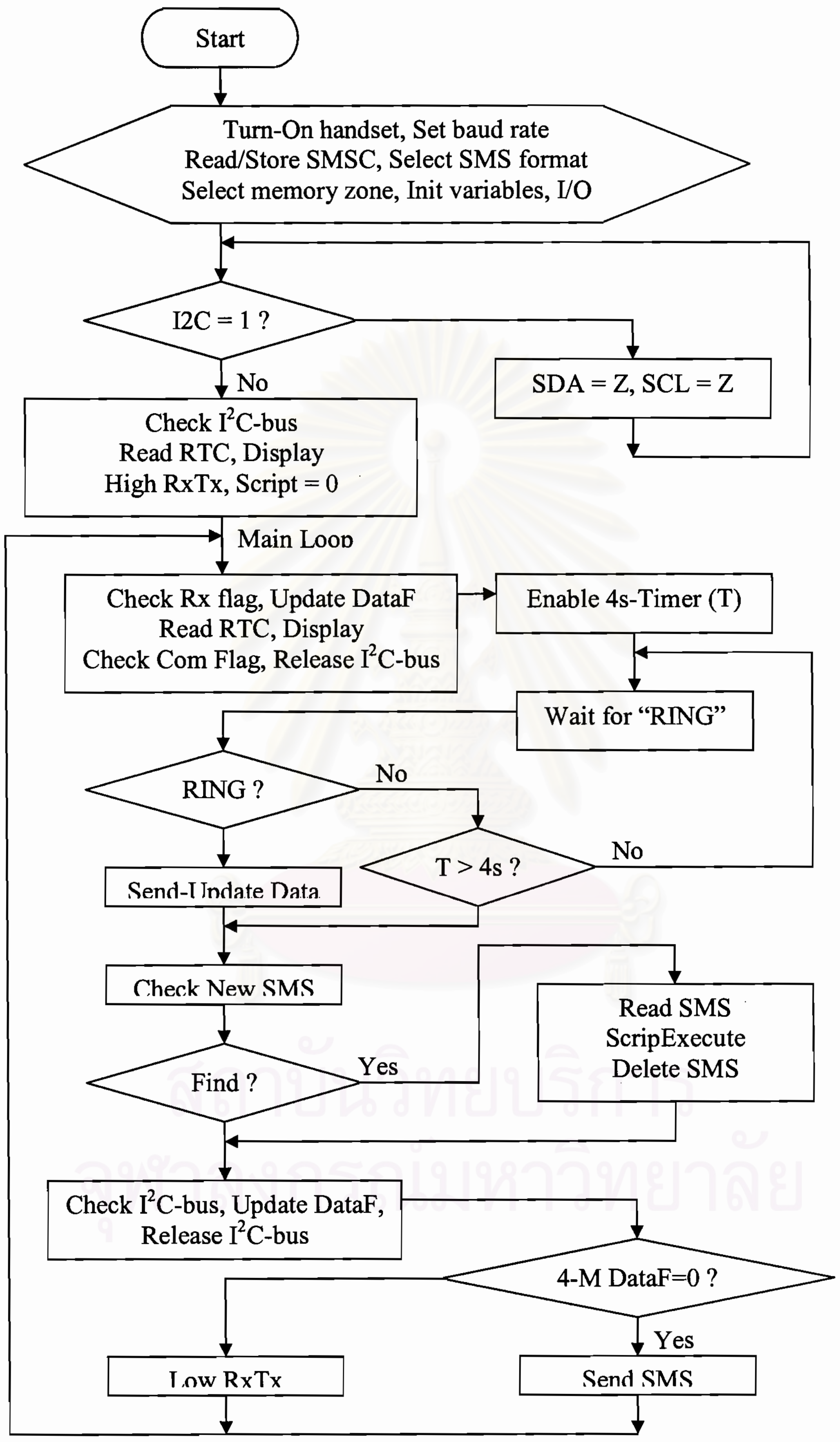


Fig 52. Radio Link Main Program Flowchart

Send SMS: For the coding process, see section 3.2.5. For this procedure (see Fig 53), the input variables are:

- Script: is a pointer to indicate which kind of SMS should be send. SMS is set according to the Script variable as follow:
    - Script.0 = 1 : Send "The memory is full at mobile node: ROBOTXXX"
    - Script.1 = 1 : Send "Critical input detected"
    - Script.2 = 1 : Send <Report> follow by "All your commands are well done"
    - Script.3 = 1 : Send SMS according to <Read> command. The format look like data in EEPROM from address $121 to $195
    - Script = 0 : Do nothing and exit procedure.

- AutoNum: indicates which destination SMS should be sent to.
    - AutoNum = "0" : Return to the same number (the one who just send a SMS to this mobile node)
    - AutoNum = "1" : Send SMS to predefined number located in the EEPROM from address $12 and $22. The SMS can be sent to just the one located from address $12 or both of predefined destination address. This option is set in EEPROM ($4). If this EEPROM value is 1, then only the destination located from address $12 can be sent to, and if it is 2, then both of them can be sent to.
    - AutoNum = "2" : Send SMS to predefined number and also the one who send us SMS. For predefined number, it depends on the value of EEPROM ($11).
- SendOption: locates in EEPROM ($11), note B0. This byte can carry the value 1 or 2. It is used to indicate which predefined number should be sent. B0 has to work with B1 (EEPROM ($21)) which is a temporary option (initially B1=0, see Fig 53)

- B0=1, B1=X : means that "Send to only one predefined number that located from address $12". This predefined number belongs to central phone number.
- B0=2, B1=0 : means that "Send to both predefined number, but now, it is time to send the first one"
- B0=2, B1=2 : means that "Send to the second predefined number located from address $22". This number belongs to manager's phone number.

If AutoNum = "2", we start to send the predefined number first and then reset AutoNum to "0", but not update to the memory (EEPROM ($4)), and then go back to the top of SendSMS function to send with AutoNum = "0". It looks like infinite loop, however, after sending SMS process, we check these variables whether to go back to the top of procedure or exit procedure (see Fig 53).

Even though this SendSMS procedure supports many options, we only use AutoNum = "0" and "1". For AutoNum = "2" is reserved for other application or future use. Any recommendations are welcome.

Script Execution: We can say that this function is a new innovation technology for sensor node. With this feature, we can order the sensor node or DTE to do some tasks as we order it. We intend to develop this feature to execute more complicated script like to schedule the tasks, not only set or clear the task that make the feedback feature more interesting. For the current version of Script Execution, see Fig 54.

This algorithm is based on string manipulation. After detecting the character "<" in the message (text format not PDU), we will read the following text (we use space as marker to separate each word) and compare it to all the predefined keyword and input/output port name. If they match each other, we will read the following text that defines the corresponding action of the keyword then execute it as it is. We scan like this until the end of the message.

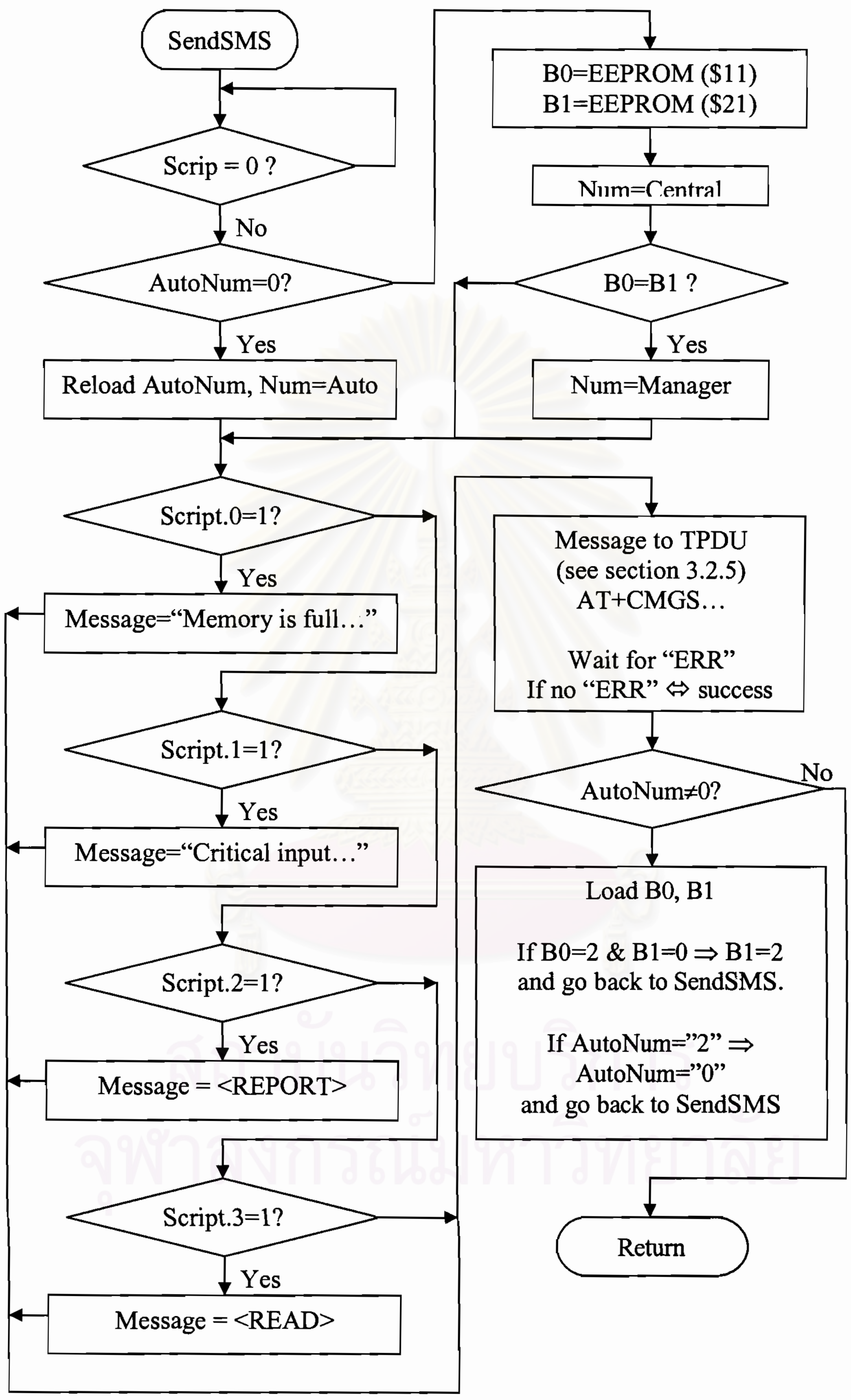
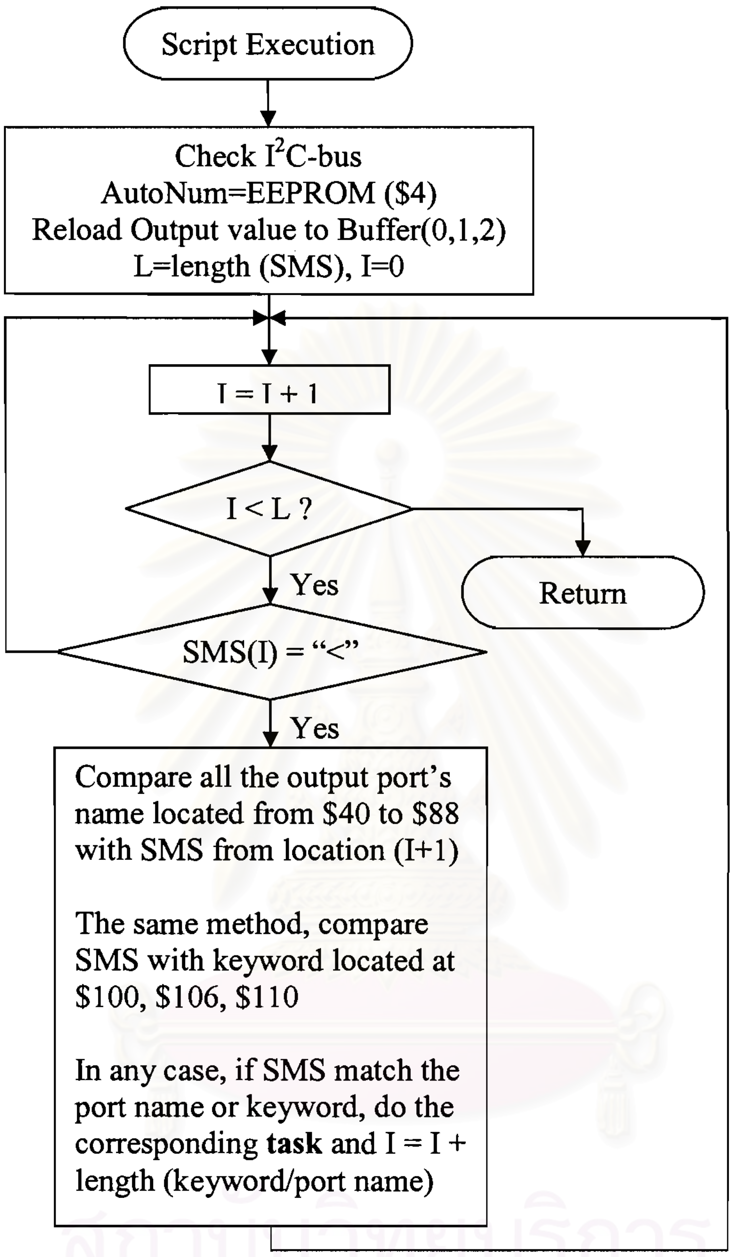


Fig 53. Send SMS Flowchart



Script Execution
Check I^2C-bus
AutoNum=EEPROM ($4)
Reload Output value to Buffer(0,1,2)
L=length (SMS), I=0
I = I + 1
I < L ?
Yes
Return
SMS(I) = "<"
Yes
Compare all the output port's name located from $40 to $88 with SMS from location (I+1)
The same method, compare SMS with keyword located at $100, $106, $110
In any case, if SMS match the port name or keyword, do the corresponding **task** and I = I + length (keyword/port name)

Fig 54. Script Execution Flowchart

Send-Update Data: This procedure ensures the data exchange between mobile node and central node. The exchange protocol is illustrated in Fig 12 whereas Fig 55 illustrates the way to implement this protocol.

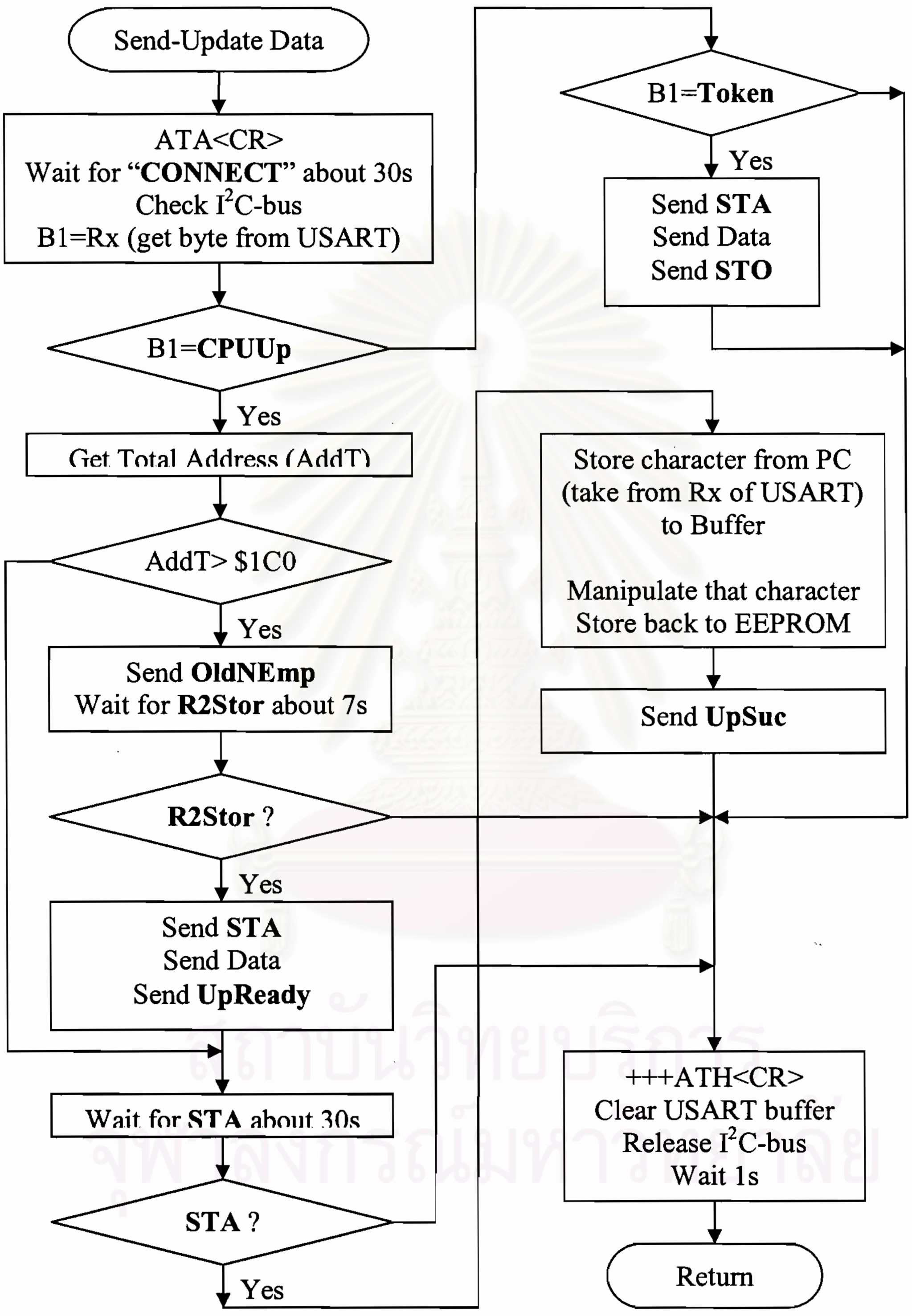


Fig 55. Send-Update Data Flowchart

3.2.7 CPU Module

For this module, besides the main program (main loop), other function/procedure such as CheckRxFlag (also has in Radio Link Module but not complicate like this one), ScanAndStoreDig, ScanAndStoreAna, SendAlarm and SendBufferOverRun are important as well. We will discuss about this function/procedure one by one.

Main Program: In this module, we also start our program by initialize some pins and variables for use in the next step (see Fig 56). At this time, we also check the present of I^2C programmer. If this connection is detected, we change the SDA and SCL line to input/high impedance (z) in order to let I^2C-bus free.

The main loop is started from this point. We begin by checking the scanning period value (T2). If it is bigger then 0, then the normal operation is selected, otherwise, jump to the next step in order to scan as fast as possible. If T2 > 0, we will go to low power mode until the rising edge of input clock is detected. Next, we check 4-MSB of DataF (EEPROM ($7)) and execute the corresponding task setting by Radio Link module (see CheckRxFlag section). After executing and updating DataF, we check the 4-LSB of DataF whether to clear the acknowledge pin (RxTx) or not. These process belong to our modify I^2C protocol. After everything is finish, we start to check T2 again. Normally, we can separate the program into 2 parts since the first test of T2, however, as each process have many things in common, and so to check every time the process is different is better. In here, if T2 = 0, it means that the security guard function is enable so, we only read the input from digital & analog port and check with the critical value to decide whether the inputs reach alarm condition or not. These processes are assured by ScanAndStoreDig and ScanAndStoreAna procedure. If T2 $\neq$ 0, the normal operation is enable so, we start to read RTC and count up the scanning period counter (I2) by 1. After this, we check whether I2=T2 or not, if it is that case, we start to read and store the value from digital and analog input to corresponding input-buffer (ScanAndStoreDig and ScanAndStoreAna). For GPS, we check first if EEPROM ($5) ="1", we read GPS data and store it otherwise this process is skipped.

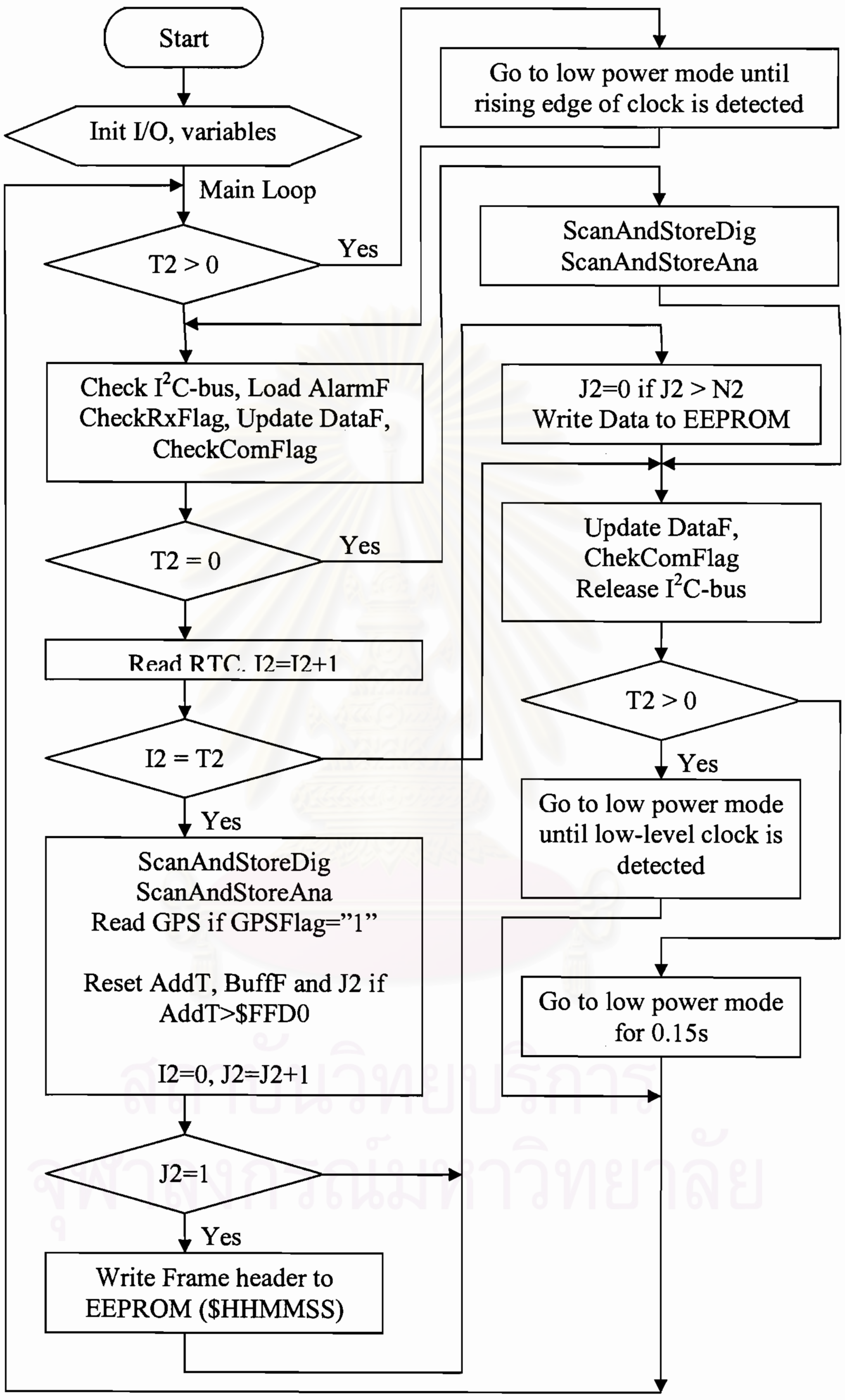


Fig 56. CPU Main Program Flowchart

Now, it is time to store the data from input-buffers to EEPROM. However, before this process start, checking the available memory is taken into account by comparing the total address counter with predefined threshold ($FFD0 was decided because we check only the address where to start storing the data not where the data is located at the last). This value is calculate and chosen with the maximum data in byte for one sampling time (2 for Digital input + 2*8 for Analog input + 10 for Latitude + 11 for Longitude + 6 for Altitude = 45 bytes) whereas the free space between $FFFF and $FFD0 is 47 bytes > 45 bytes ⇒ OK. If the total addresses counter > $FFD0 then we start to store the data from beginning ($1C0) and also reset alarm for buffer over run automatically. If everything is OK, we reset I2, count up the frame size counter (J2) by 1 and store the frame header if J2 = 1 or reset J2 if J2 >= N2 (frame size value), then store the input data from input-buffer to EEPROM byte by byte. Page write can not be used in this case because our data's length is not fix (depend on the setting), so random write mode is used, result need more time for each write process as it is recommended by Atmel company.

Finally, we update DataF to the white board (this data can be changed by ScanAndStoreDig and ScanAndStoreAna) and check the 4-LSB of DataF to see whether to release the acknowledge line (make RxTx pin HIGH) or keep it LOW. After all, if T2 > 0, we go to low power mode until falling edge or LOW state of clock line is detected, otherwise, just go to low power mode for a while (about 0.15 s) and then jump back to main loop to repeat all the step forever.

As you can see that if we omit this low power mode (don't want to save the power consumption), we can scan faster then this (about 0.1 ms) because, at that time, the factor that limits the scanning speed is the conversion time of analog to digital (20 μs for each input, so the delay for 8 inputs is about 80 μs (time for reading all the 8 digital input is 1 μs) or just say 0.1 ms. This delay can be smaller if the input impedance of analog port is smaller then 10 kΩ. From this calculus, section 2.2.2 say that the maximum scanning speed is about 0.2 s.

CheckRxFlag: These function do similar actions as CheckRxFlag function in Radio Link module. However, the task is different (see Fig 57). Instead of sending SMS, this function has to prepare the SMS string according to the type of SMS that is defined by Radio Link module.

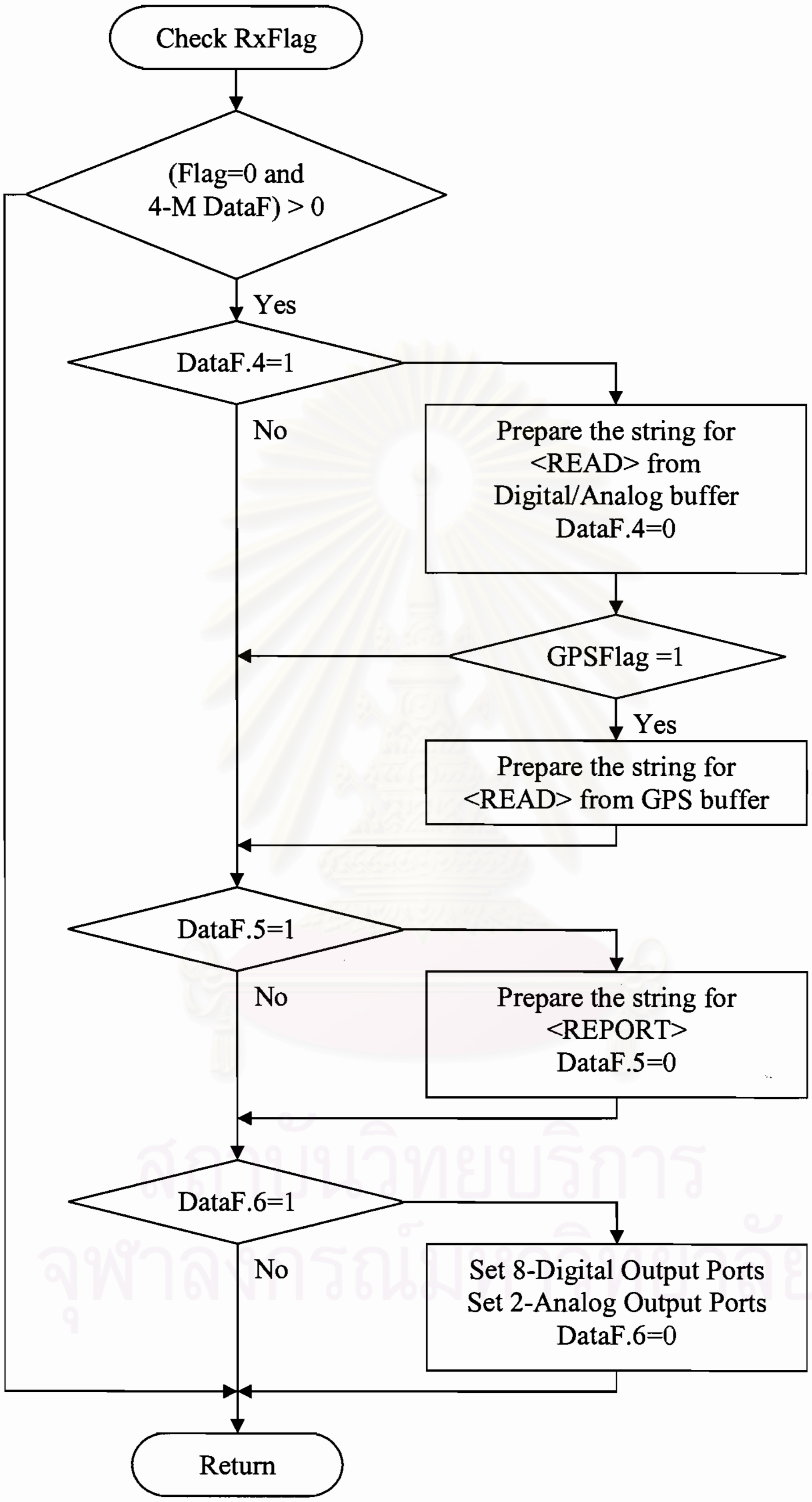


Fig 57. Check RxFlag Flowchart

This function begins by checking the acknowledge pin of Radio Link, which is Flag pin for CPU. If this pin is LOW and 4-MSB of DataF differs from 0, the analyzing process starts otherwise, we do nothing. Support that this condition is true, so we will check DataF bit by bit as below:

- DataF.4 = 1 : means that Radio Link module want CPU module to prepare the message for <READ> command. So, CPU will prepares the message for READ command with the data taking from input-buffer and then reset that bit to 0 but not yet update back to the white board. We will write it back later in byte not bit by bit. For how to prepare that message, we would skip it because we don't want go to too deep in the programming technique.
- DataF.5 = 1 : means that Radio Link module want CPU module to prepare the message for <REPORT> command. After prepare that message, CPU will reset that bit to 0. This action is interpreted by Radio Link module as a command to send SMS but if only 4-MSB of DataF is 0, SendSMS function will not run because this function needs other variable as describe in section 3.2.6.
- DataF.6 = 1 : means that Radio Link detects a script that controls the output ports. So after executing the script successfully, the Radio Link module wants CPU module to send these values (digital output state and/or analog output state) to the corresponding output ports. After receiving this message from Radio Link module, CPU module will execute this command and reset the corresponding bit to 0. For how to set the output port is too deep into programming technique, so we want to skip it here, however, section 3.1.4 and 3.1.5 can answer these questions.

See also Fig 41 for more information about DataF (EEPROM ($7)) format and message format in the EEPROM.

ScanAndStoreDig: This function will read the digital input and store in input-buffer for SMS (DataIn[0]) if the digital mask (DigMask) is bigger then 0, and compare with digital critical value (DigCrit = EEPROM ($31)). If the value from input port is the same as the value of DigCrit, we count down alarm counter (AlarmC[0]) while it still bigger then 0, otherwise alarm is activate (SendAlarm function is executed). The input port value is coded into 2 bytes, as shown in Fig 13, and store in the input-buffer (DataS). After storing, we check the total counter address (AddT) whether it is over $CCCC or not ($CCCC is about 80% of total memory), if yes, SendBufferOverRun procedure is executed and then disable Send Buffer Overrun flag because we want to send this information just only one time. This flag will be reset automatically when we start to store the data from the beginning ($1C0). See Fig 58.

ScanAndStoreAna: This function will check all the analog input mask (AnaMask) and see bit by bit. If the bit is set to 1, the corresponding analog input port will be activate and start the analog to digital conversion. After the conversion is finish, we store it in input-buffer for SMS (DataIn[1 to 8]) and then round this value to just 8-bit and then compare with the corresponding analog critical value (AnaCri). However, the comparison mode is chosen by looking at analog comparison flag (AnaComp = EEPROM ($3F)), if the correspond bit is 1, it means "Check if analog input is bigger then critical value or not" otherwise the comparison word is "Check if analog input is smaller then critical value or not". In both case, if the condition is true, SendAlarm function is executed) otherwise, count down alarm counter (AlarmC[port index]) while it is still bigger then 0. The input port value is coded into 2 bytes, as shown in Fig 13, and store in the input-buffer (DataS). After storing, we check the total counter address (AddT) whether it is over $CCCC or not, if yes, SendBufferOverRun procedure is executed and then disable Send Buffer Overrun flag. See Fig 59.

SendAlarm: This function will check whether alarm counter (AlarmC) is still smaller then alarm value (critical value of the corresponding port (digital and/or analog)) or not, if yes, we just count up AlarmC by 1, otherwise we continue to check whether alarm sending by SMS flag (AlarmF) is enabled or not, if AlarmF is enabled (AlarmF = "1"), we reset AlarmC and set the corresponding bit of DataF to 1 for Radio Link to see that CPU module has message to send out. Finally, exit function (see Fig 60).

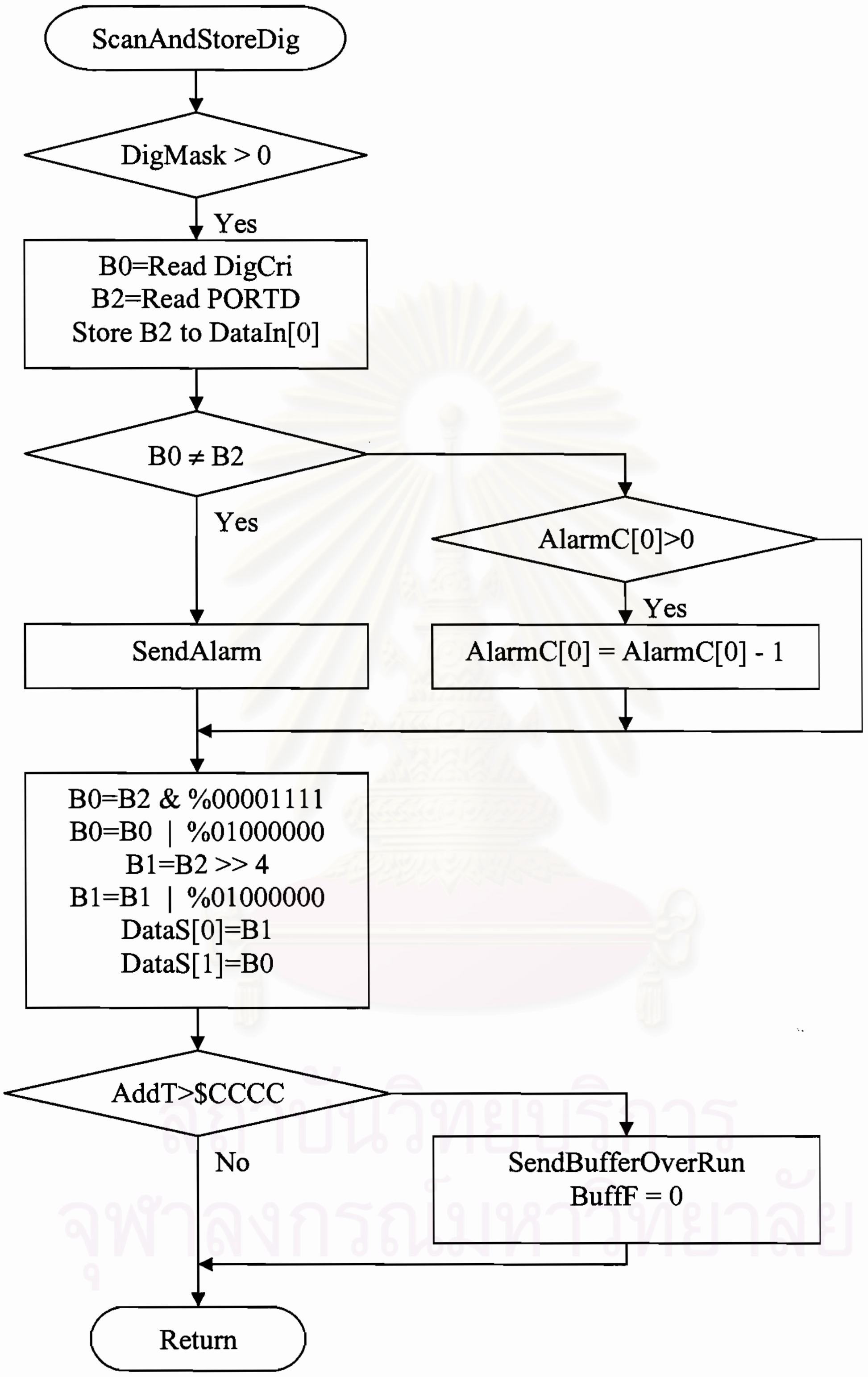


Fig 58. Scan and Store Digital Input Port Flowchart

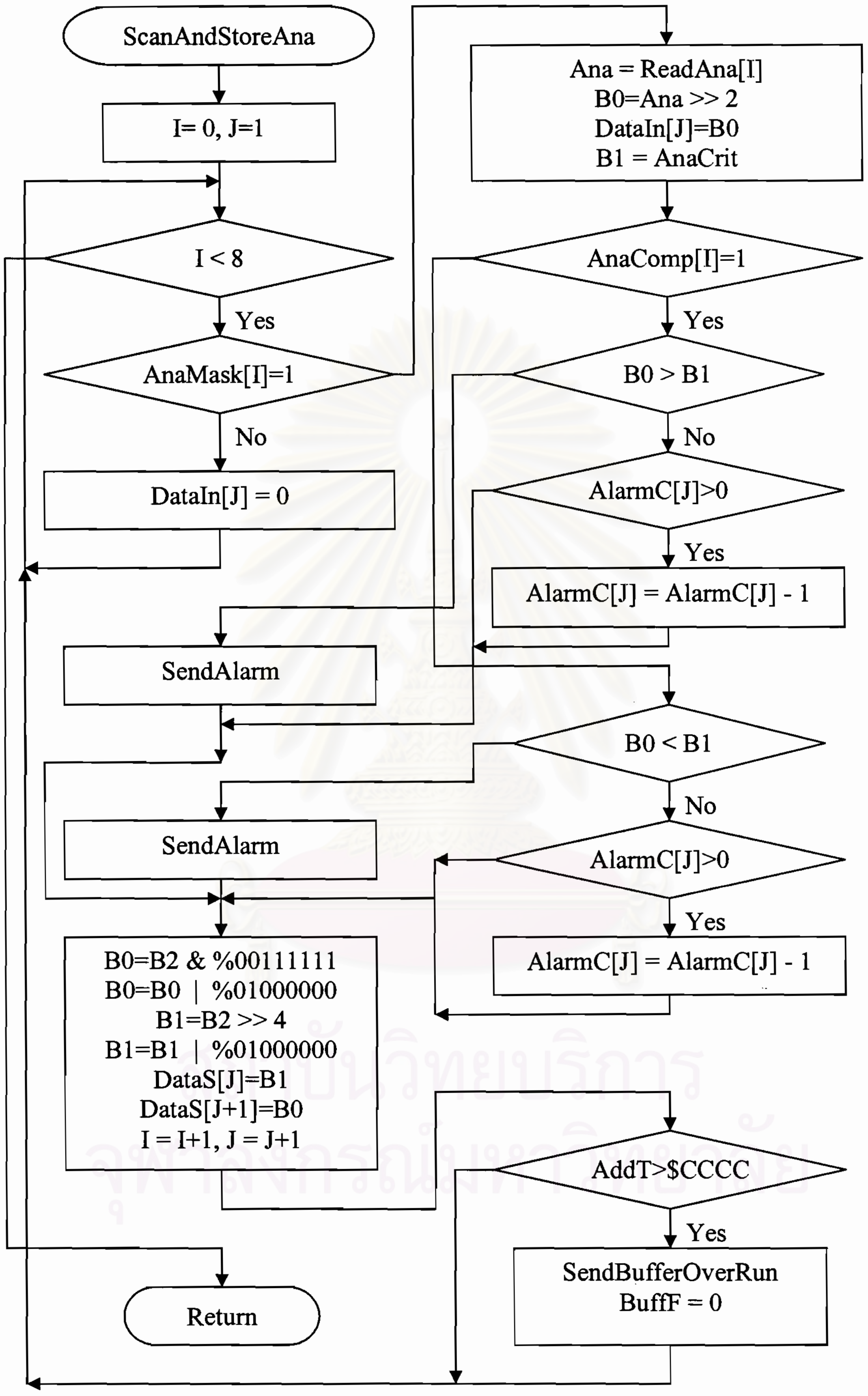


Fig 59. Scan and Store Analog Input Port Flowchart

SendBufferOverRun: For this function, we just check whether alarm sending by SMS flag (AlarmF) is enabled or not. If it is enabled, we will set the corresponding bit of DataF to 1 and exit the function simply (See Fig 61). Fig 62 illustrates CheckI^2C-bus algorithm.

Now, you can see how the mobile node works and can imagine how to react to the event coming from input ports or mobile phone, hence we will leave the explanation right here.

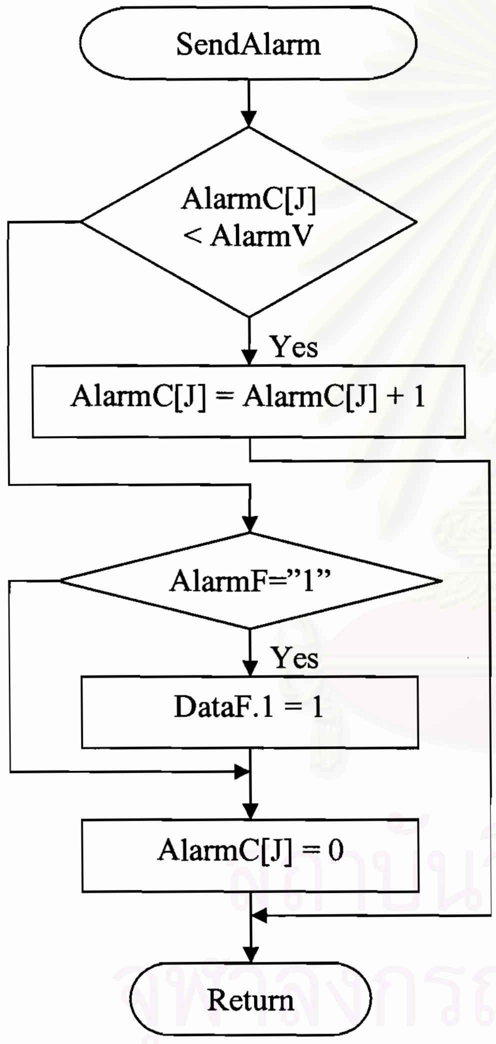


Fig 60. Send Alarm Message Flag Flowchart

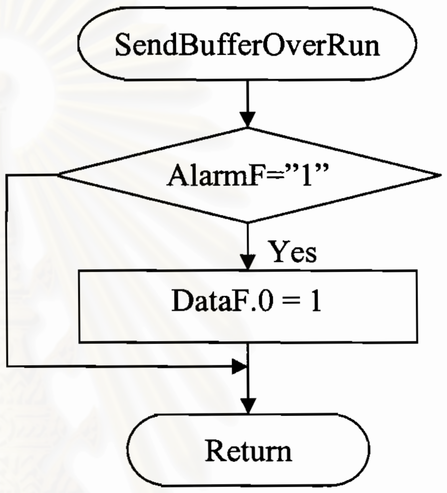
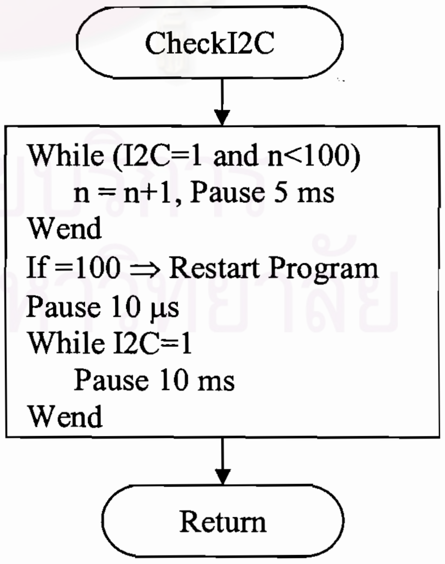


Fig 61. Send Buffer Overrun Flag Flowchart



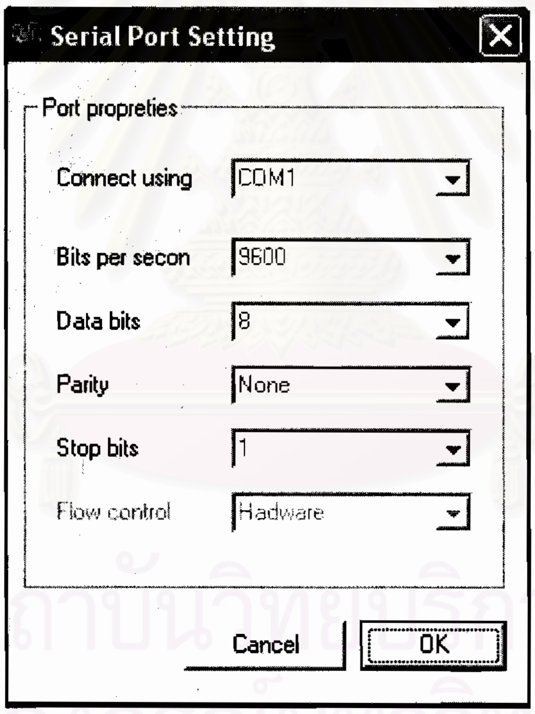
```
While (I2C=1 and n<100)
    n = n+1, Pause 5 ms
Wend
If =100 ⇒ Restart Program
Pause 10 µs
While I2C=1
    Pause 10 ms
Wend
```

Fig 62. Check I^2C-bus Flowchart

3.3 Software Flexibility and Utilization

As you can see clearly about the functionality of mobile node, in this section, we would like to present you our complete program for the central node. This program is named as "Central of Tele-measurement Sensor Network". It has built-in database and other functionality as mention in chapter 2, section 2.2.2.

Each forms and some command buttons, check boxes, option buttons, list boxes, combo boxes, driver list boxes, directory list boxes, file list boxes, scroll bars, text boxes and labels will be given a short description and explanation on how to use it. In this program, we divide it into 8 forms according to its functions. Let see one by one, start from the overview of this program to see how it looks like. Fig 63 illustrates this.


Serial Port Setting

Port propreties

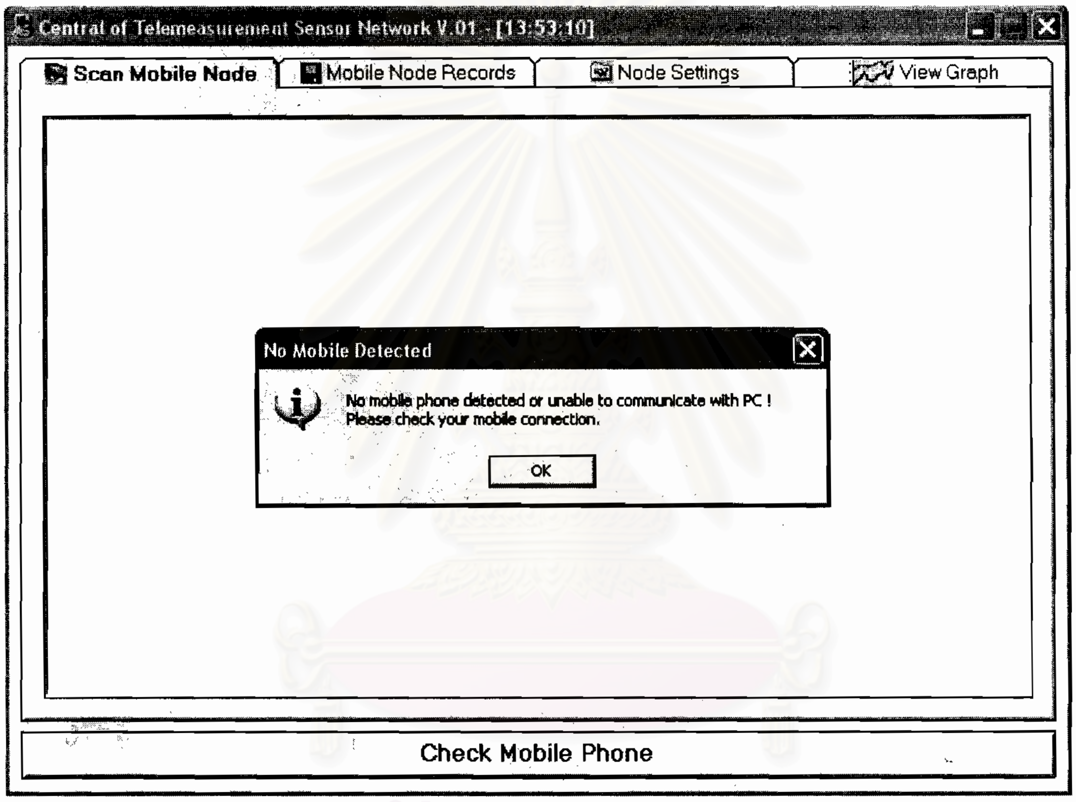
Connect using	COM1
Bits per secon	9600
Data bits	8
Parity	None
Stop bits	1
Flow control	Hadware

Fig 63. Serial Port Setting Form

This form allows the user the ability to select the communication port and configure its setting. Automatically, this form will show all the available COM port supported by your computer. For the question "How to do this?" we can only say it is out of scope for this document and it is too deep in to programming level that is difficult to be explained in brief like this.

After choosing your corresponding COM port and appropriate setting such as baud rate, data bits, parity and stop bit from the corresponding combo box, just click **OK**, we will do the rest for you.

After click **OK** button, this form will we be hidden and unloaded from the memory in order to free the memory for other application. However, before this form is unloaded, it will load the main form as show in Fig 64.



No Mobile Detected

No mobile phone detected or unable to communicate with PC ! Please check your mobile connection.

OK

Fig 64. Central of Tele-measurement Sensor Network Main Form

This form consists of 4-Tab menus named as **Scan Mobile Node**, **Mobile Node Records**, **Node Settings** and **View Graph**. As its name, each Tab menu has it own functionality. In here, a message box display that no mobile phone is detected because, in fact, we do not connect the mobile phone with the PC via COM1 as it inspects. We do this because we want to show that our program is build with many intelligent artificial algorithms. However, we do no guaranty that this program will run with no error because, as during testing period, no error is found, we also can not imagine all the possibility that error can come out, hence no error detection/correction algorithm is implemented. This message box will not appear if the mobile phone is connected to the PC via a correct port chosen by the first form.

Scan Mobile Node: This Tab is use to display any information concerning the store/update process with the process's time then mobile node name, start/stop time and the result of the process for each mobile node as show below:

```
samedi, 12-mars-2005 at 14:39:21
Update Start >>>
ROBOT003 >> START 14:39:22 => STOP 14:40:15    ERROR: NETWORK IS BUSY
ROBOT008 >> START 14:40:17 => STOP 14:41:10    ERROR: NETWORK IS BUSY
ROBOT001 >> START 14:41:12 => STOP 14:42:06    ERROR: NETWORK IS BUSY
ROBOT002 >> START 14:42:08 => STOP 14:43:05    ERROR: NETWORK IS BUSY
ROBOT005 >> START 14:43:07 => STOP 14:44:03    ERROR: NETWORK IS BUSY
ROBOT004 >> START 14:44:05 => STOP 14:44:58    ERROR: NETWORK IS BUSY
ROBOT006 >> START 14:45:00 => STOP 14:45:53    ERROR: NETWORK IS BUSY
ROBOT009 >> START 14:45:55 => STOP 14:46:48    ERROR: NETWORK IS BUSY
ROBOT010 >> START 14:46:50 => STOP 14:47:43    ERROR: NETWORK IS BUSY
ROBOT007 >> START 14:47:45 => STOP 14:48:38    ERROR: NETWORK IS BUSY
End Process
```

This information will be saved automatically while this program is turn off. The name is generated automatically with the date-time information (for example: 2005-03-12--14-39-21.rtf) whereas the path of this file is the default part set by the user (see **Node Setting** Tab). This error information come from the first test (no mobile phone is connected).

Mobile Node Records: This Tab is used to record the mobile node information into database. It allow the user to add, edit, save, delete the record (record means the information corresponding to each mobile node) and also allow the user to print the data corresponding to each record or all the record. The **Print** button will load the **Print Record** form for you and **Find** button will load **Find the Record** form for you. See **Print Record** form and **Find the Record** form. Fig 65 illustrates this Tab.

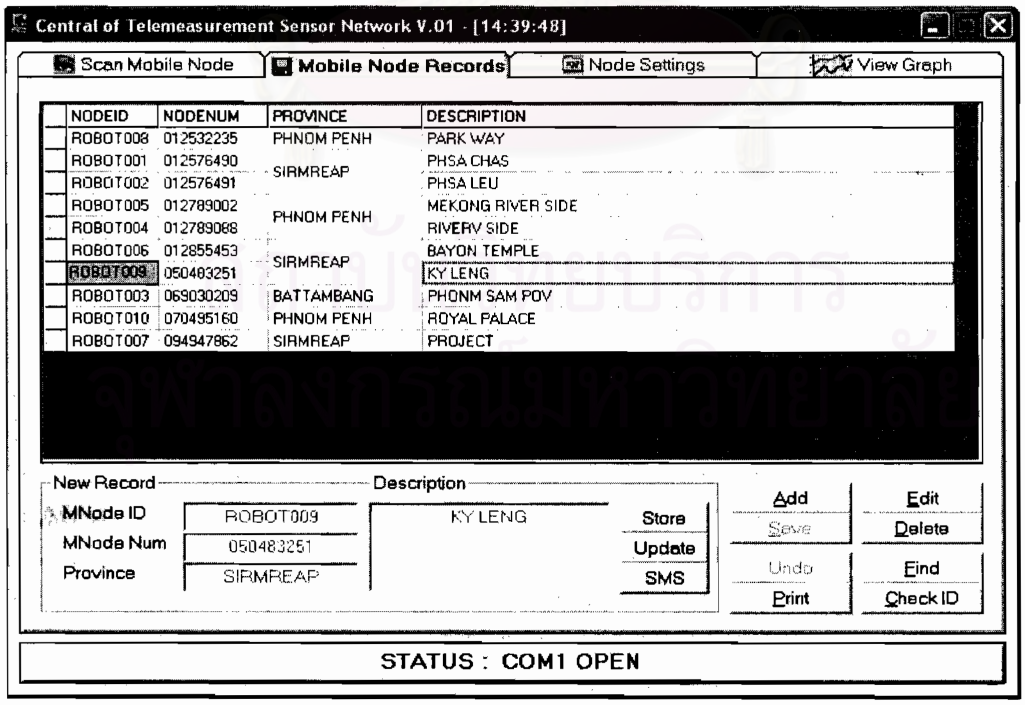
By clicking on a cell of the Flex Grid (look like a table that bound to the database, located on the top of the form), the corresponding information will be shown on the text box (*MNode ID*, *MNode Num*, *Province* and *Description*) and the *NODE ID* cell is also highlighted with clear green background to attract attention of the user.

Store button is used to connect to the mobile node (the one that is displayed on *MNode ID* text box) to retrieve the data. In Fig 65, if we click on **Store** or **Update** or **SMS** button, the corresponding mobile node will be **ROBOT009.**

Update button is used to retrieve data from the corresponding mobile node as **Store** button, however, after received all the data, the PC will send back the new setting data to that mobile node.

SMS button is used to send a script command to the corresponding mobile node. It will load **Send SMS** form for you. See **Send SMS** form.

Check ID button is used to verify whether the mobile node that is connecting to our PC (while programming) is belonged to our group or not because that mobile node can be our product but is not used with this project. By a simple click on this button, our program will read the corresponding information from EEPROM and compare it to all the existing mobile node IDs in the database. If we can not find any ID match this information, it means that this mobile node does not belong to this group/project; otherwise, we will display all its information on the text box.



NODEID	NODENUM	PROVINCE	DESCRIPTION
ROBOT008	012532235	PHNOM PENH	PARK WAY
ROBOT001	012576490	SIRMREAP	PHSA CHAS
ROBOT002	012576491		PHSA LEU
ROBOT005	012789002	PHNOM PENH	MEKONG RIVER SIDE
ROBOT004	012789088		RIVERV SIDE
ROBOT006	012855453	SIRMREAP	BAYON TEMPLE
ROBOT009	050483251		KY LENG
ROBOT003	069030209	BATTAMBANG	PHONM SAM POV
ROBOT010	070495160	PHNOM PENH	ROYAL PALACE
ROBOT007	094947862	SIRMREAP	PROJECT

New Record

MNode ID: ROBOT009

MNode Num: 050483251

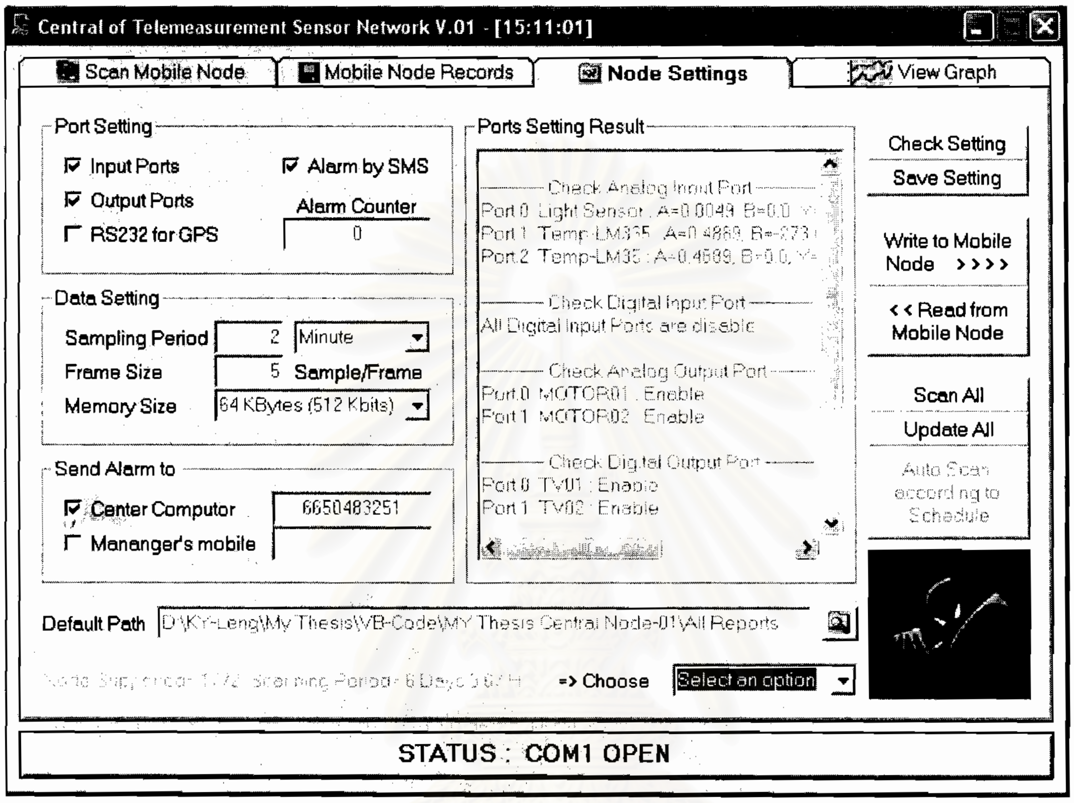
Province: SIRMREAP

Description: KY LENG

STATUS : COM1 OPEN

Fig 65. Mobile Node Records Tab

Node Settings: This Tab is used to set the mobile node setting or we can say that it is used to set the project properties. It consists of 4 frames (Port Setting, Port Setting Result, Data Setting and Send Alarm to) and many things more.



Port Setting
- ☑ Input Ports
- ☑ Output Ports
- ☐ RS232 for GPS
- ☑ Alarm by SMS
- Alarm Counter: 0

Data Setting
- Sampling Period: 2 Minute
- Frame Size: 5 Sample/Frame
- Memory Size: 64 KBytes (512 Kbits)

Send Alarm to
- ☑ Center Computor: 6650483251
- ☐ Menanger's mobile

Ports Setting Result

```
---------- Check Analog Input Port ----------
Port 0  Light Sensor : A=0.0049, B=0.0, Y=
Port 1  Temp-LM335 : A=0.4889, B=-273
Port 2  Temp-LM35 : A=0.4889, B=0.0, Y=

---------- Check Digital Input Port ----------
All Digital Input Ports are disable

---------- Check Analog Output Port ----------
Port 0  MOTOR01 : Enable
Port 1  MOTOR02 : Enable

---------- Check Digital Output Port ----------
Port 0  TV01 : Enable
Port 1  TV02 : Enable
```

Default Path: D:\KY-Leng\My Thesis\VB-Code\MY Thesis Central Node-01\All Reports

Node Supported: 1/72 Scanning Period: 6 Days 5 6/ H =>Choose: Select an option

STATUS : COM1 OPEN

Fig 66. Node Settings Tab

Port Setting frame: is used to group the option related to input/output port of the mobile node. It consists of:

- *Input Ports* check box: is used to enable or disable the input port of mobile node. If it is checked, the **Input Port Setting** form will be loaded for you, so you can give a name to a physical port name for you can recognize it easily. See **Input Port Setting** form for more detail.
- *Output Ports* check box: is used to enable or disable the output port of mobile node. The same as for *Input Ports* check box, if it is check, the **Output Port Setting** form will be loaded and then you can give each physic port a virtual name, enable or disable, etc. See **Output Port Setting** form for more detail.
- *RS232 for GPS* check box: is used to enable or disable the RS-232 port of mobile node. If it is checked, the mobile node will

read the GPS coordinate and store it as other input data; otherwise, the mobile node will leave this RS-232 port free.

- *Alarm by SMS* check box: is use to enable or disable alarm by SMS option. If it is checked, the mobile node will report any unpleasant condition by SMS to the central node (PC that run this program) and/or manager mobile phone. That message can be "Critical input detected at mobile node" or "The memory is full at mobile node: ROBOTXXX".
- *Alarm Counter* text box: is used to set the upper bound counter limit for the mobile node while critical input is detected. See chapter 2 and Fig 60 for more detail. This value is named as AlarmV in the program at mobile node. The value 0 means that if the input reaches the critical value, alarm by SMS will be activated directly no need to count how many time the input value is criticize.

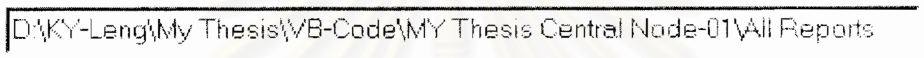
Port Setting Result frame: is used to display the status of enable port with its virtual port name, physical port name, coefficients, alarm level, etc. As you can see on the text box inside this frame, it display both input and output status. The two scroll bars is used to slide the text box to see the information that can not be displayed.

Data Setting frame: is used to group the scanning/storing process properties. It consists of:

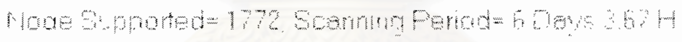
- *Sampling Period* text box/combo box: is used to set the sampling period value on the mobile node. It specifies the mobile node how often mobile node has to scan the input and store that data to the memory.
- *Frame Size* text box: is used to set the frame size value on the mobile node. It indicates to the mobile node how often mobile node has to add the frame header, containing the synchronizing frame header and time information, to the data stream. See Fig 13 for more information about the format of frame header
- *Memory Size* combo box: is used to calculate the scanning frequency for the central node. This property is used by central node only, not for mobile node. It will determine how long we can leave the mobile node to store the data and so we can know when to retrieve the data from mobile node. See **Check Setting** button.

Send Alarm to frame: is used to group the wireless link property. It contains the central computer phone number and manager phone number. The central computer check box is automatically checked when you fill its text box with a phone number. If you fill both text boxes, it means that you want mobile node to send SMS alarm to both numbers. If you want just only one, please fill the first text box, it is taken as a priority number.

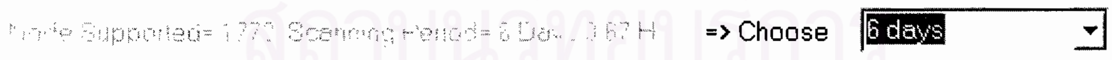
Default Path text box: is used to store the location where to store the retrieve data and report. By clicking on this button located next to the *Default Path* text box, the Folder View form will be loaded and then you can select where you want the data and report to be stored. For this setting the default path is:


D:\KY-Leng\My Thesis\VB-Code\MY Thesis Central Node-01\All Reports

Check Setting button: is used to calculate how many mobile nodes can be supported by this system and how long we can let the mobile node keep storing the data. Just a simple click on it, the result will be display on a label just in front of *Choose* combo box. For this setting, the result from calculation is:

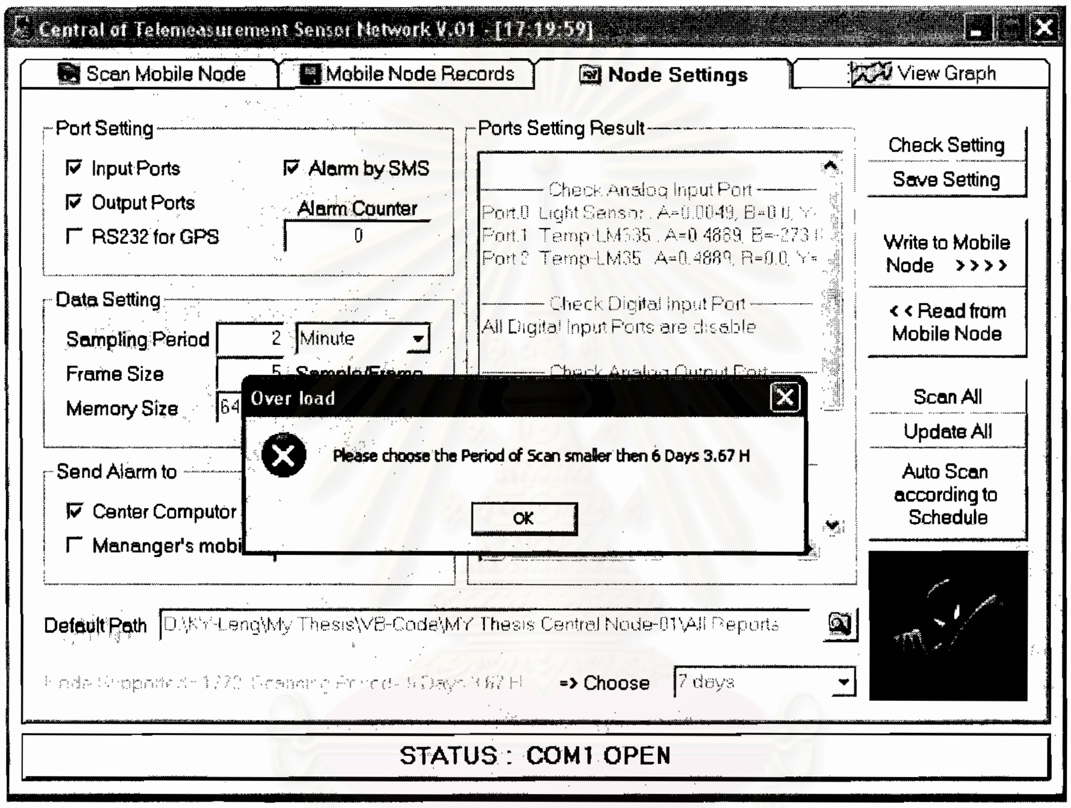

Node Supported= 1772, Scanning Period= 6 Days 3.67 H

From this information, we can decide how often central node have to retrieve the data from mobile node. To set this value to the central node, we just choose one value from *Choose* combo box located next to this label. For this setting, the longest time for the central node to retrieve data is 6 days as show below:


Node Supported= 1772, Scanning Period= 6 Days 3.67 H => Choose 6 days

Suppose we choose to scan every 7 days or other value bigger than 6 days, after choosing this value from Choose combo box, a message box, as shown in Fig 67, will be load to tell you that you can not choose this value for retrieve the data from the mobile node for this system. Auto Scan according to Schedule button will be enabled only when the correct value is chosen from this combo box.

Note that, if the settings (number of input port, scanning period, frame size) are not yet controlled by **Check Setting** button, **Save Setting**, **Write to Mobile Node, Scan All**, **Update All**, **Auto Scan according to Schedule** button are not active. To active these buttons, you can read the setting directly from mobile node, if it is connected to the PC via "*RS232 to I^2C Debug & Programming Board debug and programming board*", or by clicking on **Check Setting** button.



☑ Input Ports ☑ Alarm by SMS
☑ Output Ports
☐ RS232 for GPS
Alarm Counter: 0

Sampling Period: 2 Minute
Frame Size: 5 Sample/Frame
Memory Size: 64

☑ Center Computor
☐ Mananger's mobi

Check Analog Input Port
Port.0 Light Sensor , A=0.0049, B=0.0, Y
Port.1 Temp-LM335 , A=0.4889, B=-273.
Port.2 Temp-LM35 , A=0.4889, B=0.0, Y=

Check Digital Input Port
All Digital Input Ports are disable

Check Analog Output Port

Over load

Please choose the Period of Scan smaller then 6 Days 3.67 H

Default Path: D:\KY-Leng\My Thesis\VB-Code\MY Thesis Central Node-01\All Reports

Node Supported=1??? Scanning Period= 6 Days 3.67 H => Choose 7 days

STATUS : COM1 OPEN

Fig 67. Over Load Message Box

Save Setting button: After checking the setting of this system, we can save it by clicking on this button, so next time, when you run this program; these values will be set automatically at the right place.

Write to Mobile Node button: is used to save these setting to the mobile node while the mobile node is connected to the PC. This button is very useful during system setting. We can initialize the mobile node to recognize our system only by this button. Transparently, this button will copy the date-time information from the central node to the mobile node (into RTC register) automatically.

Read from Mobile Node button: is used to read the system setting from the mobile node while it is connected to the central node via "*RS232 to I^2C Debug & Programming Board debug and programming board*". See Fig 43.

Scan All button works the same ways as **Store** button, however, with only one click on this button, all the mobile node will be connected and retrieve the data by the central node.

Update button is used to retrieve data from all mobile nodes as **Scan All** button, however, after received all the data, the PC will send back the new setting data to those mobile nodes.

Auto Scan according to Schedule button is used as **Scan All** button, however, this retrieve action is activated automatically every scanning period chosen from *Choose* combo box. Without any intervention from operator, scanning and storing process will be made automatically.

View Graph: This Tab is used for displaying the result that we retrieve and save to the hard disk of central node. If the user don't like this interface provide by our program, they can use Microsoft Excel to plot the data as their need. The data is saved in a format that can be read from Microsoft Excel without any modification. Just want to tell you that if the user wants to see the evaluation of each input port with the time, our tool provided by this program is more powerful. Fig 68 shows the overview of this Tab. This Tab consists of 3 frames, each frame groups a group of functionality as describe below.

Select Data to Plot frame: consists of:

- *Driver* List box: is used to display all the existing drive (Fixed drive such as drive C or drive D on the hard disk; or Removable drive such as floppy drive, Zip drive or USB Flash drive; or CD-ROM drive, etc…) on the computer. From the default folder, where we store the data, drive D is selected.
- *Directory* List box: is used to display all the existing folder in the selected drive. From the default folder, where we store the data, "KY-Leng\My Thesis\VB-Code\MY Thesis Central Node-01\All Reports" is selected.

- *File* List box: is used to display all existing file in the selected folder. From the default folder, we find only one file, which is ROBOT007.gra.

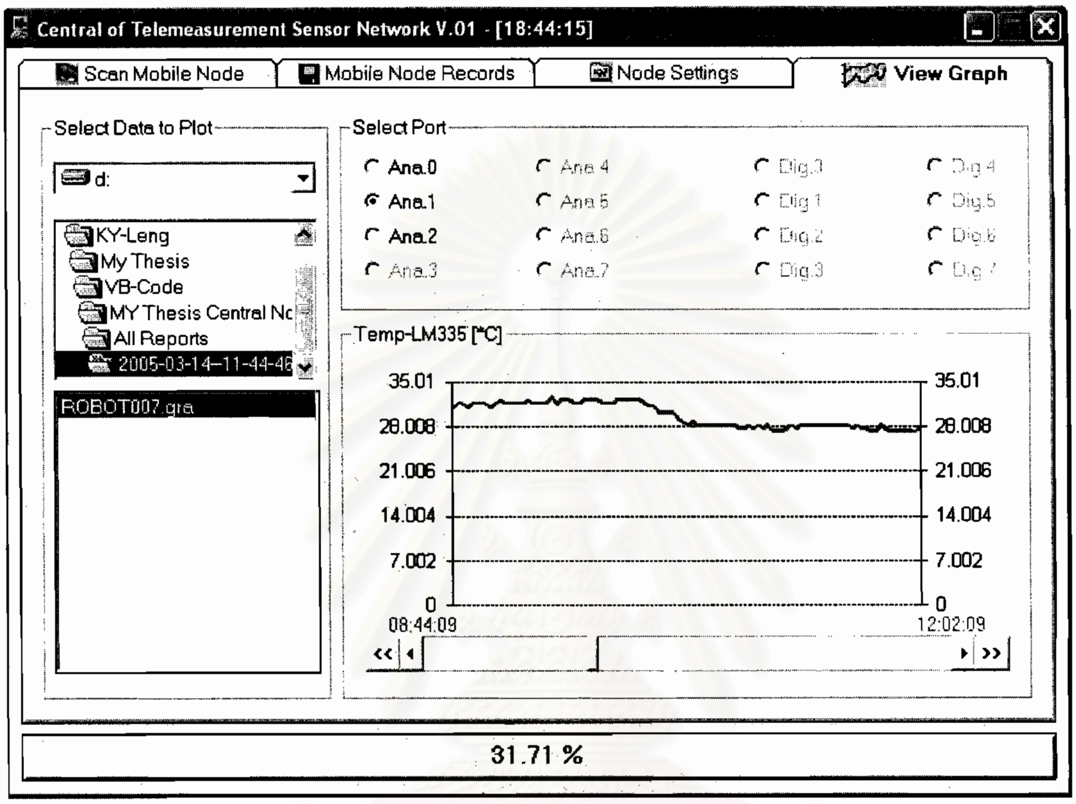


Fig 68. View Graph Tab

Select Port frame: is used to give the user facility to plot any data from the corresponding physical port. In this frame, only the physical ports are display because the virtual port name is some time too long to display. However, when you move the mouse on any physical port, the corresponding virtual port name is display as ToolTipText (The ToolTipText is a brief description of a control that appears when the user holds the mouse pointer over the current Toolbox page without clicking). Moreover, after clicking on any option button of the input port, the virtual port name will we display on the frame in where the corresponding data will be plotted (see Fig 68). Initially, all the option button are disabled to indicate that no data to display, so the user have to select a file on file list box before plotting any data.

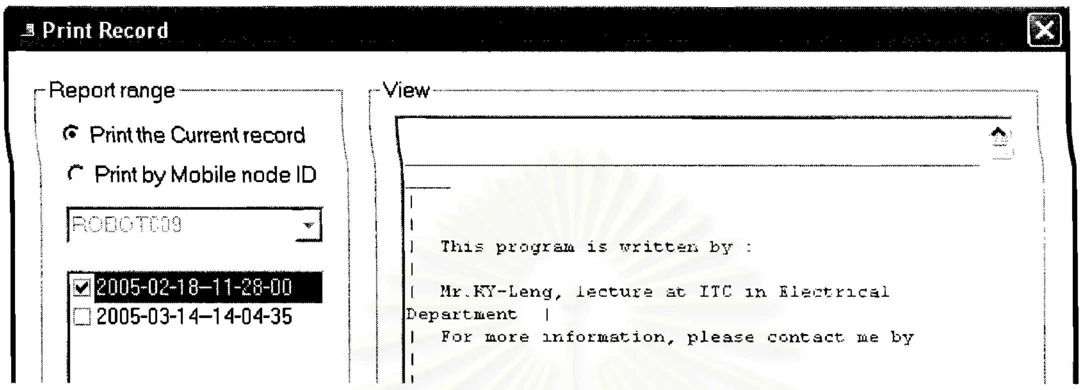
While clicking on the file (example ROBOT007.gra), our program will read all the setting saving along with the data to enable only the port that are enable in the system. This feature is very useful because we can use this program to plot other data even the setting of their system are different from the current setting supported by our system.

After select on the file, we can now start to plot the data of any enable input port by just clicking on the option button of that input port. Fig 68 shows the data of analog input port number 1 (Ana.1) drawing by our program. The corresponding virtual port name is "Temp-LM335" follow by its unit "[*C]". Below the graph, a horizontal scroll bar is used to slide the graph step by step or jump directly to some where but manually whereas other two buttons << and >> are used to slide the graph automatically. After clicking on any of these buttons, the slide will move automatically according to the direction as indicated by these buttons until the end. During this mode, the user can also stop it by clicking on ||. This button comes from one of those two button. Suppose Fast Forward button >> is pressed, automatically the Fast Backward button << will change to Pause button ||.

Print Record form: This form is used to help user to print the data by listing all the existing report in a check list box as show in Fig 69. By default, when the user click on **Print** button, the option as show in Fig 69-A is selected (Print the Current Record), however, the user can change it to another option (Print by Mobile Node ID) by just clicking on this new option and select a mobile node ID from a combo box located just below this option button. To view a file, just click on its name from the list box. To print, the user select any files they want to print by checking the corresponding name, so it means that the user can select more then one file and print it just only one time by simply click on **OK** button.

Setup Printer button is used to setup the printer before we can print this file, otherwise, a default printer will be selected to do this job. It is the same as other application, so no need to explain more then this.

Find the Record form: is used to help the user to find the mobile node information. It allows the user to select what kind of information they know from mobile node, so they can use this information as a keyword to search the complete information. We provide 3 kinds/categories of keyword such as search by *Mobile Node ID*, search by *Mobile Node Number* and search by *Province*. After selecting a keyword type from *Find By* combo box, the user can input the keyword into *Find What* text box and then click on **Find** button; the rest will be carry by our program.



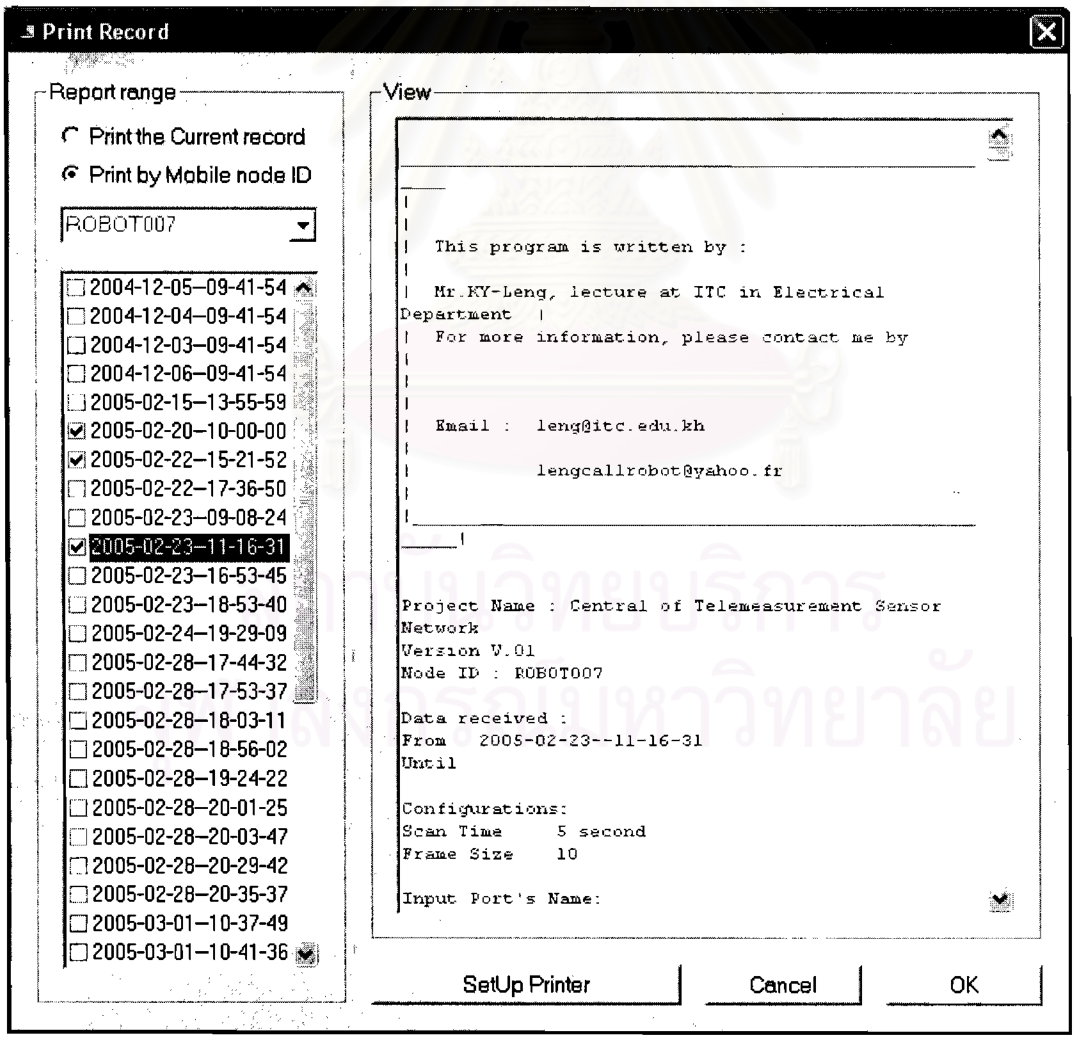
[✓ Print the Current record] Print by Mobile node ID

ROBOT009

- [✓] 2005-02-18-11-28-00
- [ ] 2005-03-14-14-04-35

```
|
|
|  This program is written by :
|
|  Mr.KY-Leng, lecture at ITC in Electrical
Department  |
|  For more information, please contact me by
|
|
```

Fig 69-A. Print Current Record Form



Print the Current record [✓ Print by Mobile node ID]

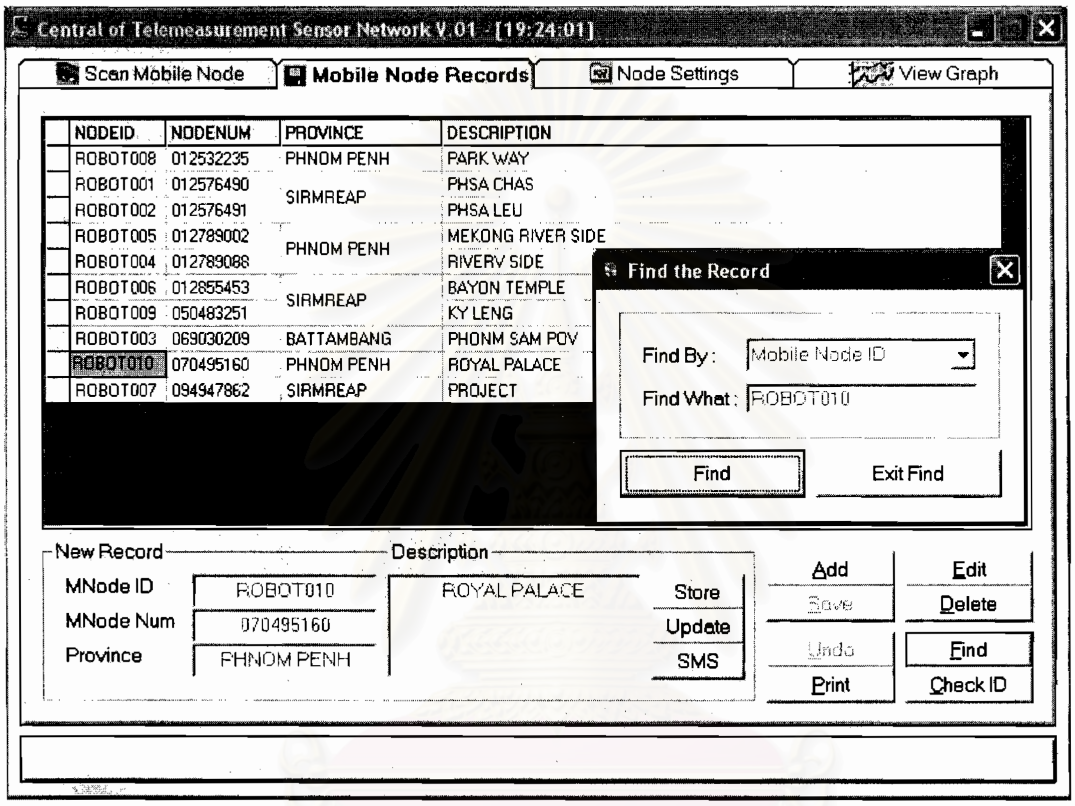
ROBOT007

- [ ] 2004-12-05-09-41-54
- [ ] 2004-12-04-09-41-54
- [ ] 2004-12-03-09-41-54
- [ ] 2004-12-06-09-41-54
- [ ] 2005-02-15-13-55-59
- [✓] 2005-02-20-10-00-00
- [✓] 2005-02-22-15-21-52
- [ ] 2005-02-22-17-36-50
- [ ] 2005-02-23-09-08-24
- [✓] 2005-02-23-11-16-31
- [ ] 2005-02-23-16-53-45
- [ ] 2005-02-23-18-53-40
- [ ] 2005-02-24-19-29-09
- [ ] 2005-02-28-17-44-32
- [ ] 2005-02-28-17-53-37
- [ ] 2005-02-28-18-03-11
- [ ] 2005-02-28-18-56-02
- [ ] 2005-02-28-19-24-22
- [ ] 2005-02-28-20-01-25
- [ ] 2005-02-28-20-03-47
- [ ] 2005-02-28-20-29-42
- [ ] 2005-02-28-20-35-37
- [ ] 2005-03-01-10-37-49
- [ ] 2005-03-01-10-41-36

```
|
|
|  This program is written by :
|
|  Mr.KY-Leng, lecture at ITC in Electrical
Department  |
|  For more information, please contact me by
|
|
|
|  Email :  leng@itc.edu.kh
|
|           lengcallrobot@yahoo.fr
|
|___________________________________________
_____|


Project Name : Central of Telemeasurement Sensor
Network
Version V.01
Node ID : ROBOT007

Data received :
From   2005-02-23--11-16-31
Until

Configurations:
Scan Time     5 second
Frame Size    10

Input Port's Name:
```

Fig 69-B. Print Record by Mobile ID Form

If the keyword matches a record, it will highlight that *NODE ID* cell and update this information to all text boxes (MNode ID, MNode Num, Province and Description). If the keyword belongs to Province type, the first one that matches the record will be display and if the user continues to click on **Find** button again, the next record that matches this keyword will be displayed, and so on. See also Fig 70.



NODEID	NODENUM	PROVINCE	DESCRIPTION
ROBOT008	012532235	PHNOM PENH	PARK WAY
ROBOT001	012576490	SIRMREAP	PHSA CHAS
ROBOT002	012576491		PHSA LEU
ROBOT005	012789002	PHNOM PENH	MEKONG RIVER SIDE
ROBOT004	012789088		RIVERV SIDE
ROBOT006	012855453	SIRMREAP	BAYON TEMPLE
ROBOT009	050483251		KY LENG
ROBOT003	069030209	BATTAMBANG	PHONM SAM POV
ROBOT010	070495160	PHNOM PENH	ROYAL PALACE
ROBOT007	094947862	SIRMREAP	PROJECT

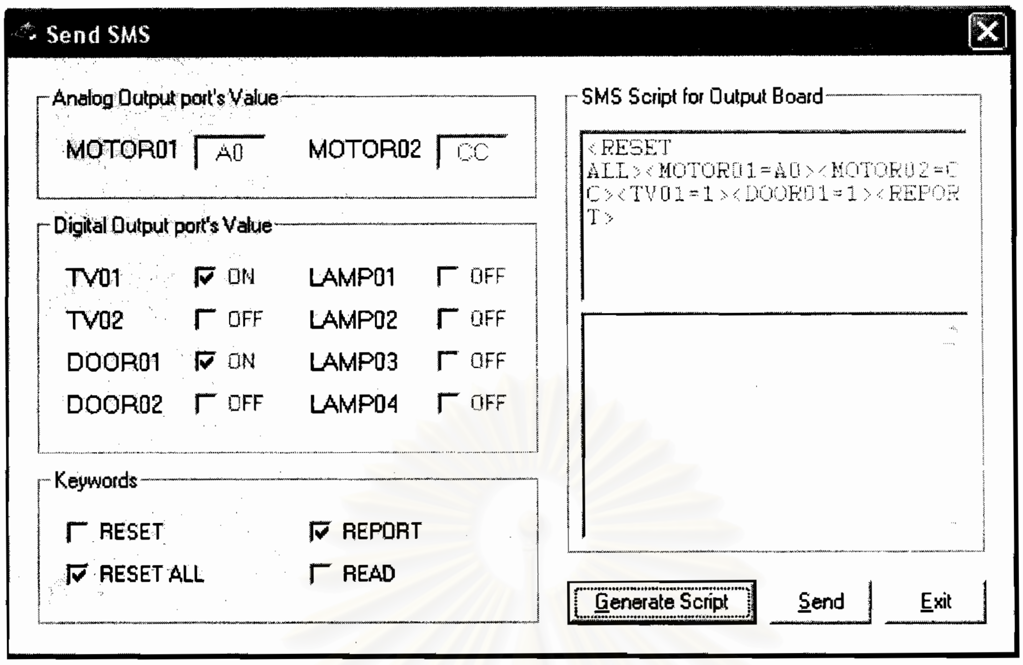
Find By: Mobile Node ID
Find What: ROBOT010

New Record
MNode ID: ROBOT010
MNode Num: 070495160
Province: PHNOM PENH
Description: ROYAL PALACE

Fig 70. Find the Record Form

Send SMS form: is used to give user the facility to send the SMS directly from computer, no need to type the script by telephone keyboard, which is not convenient. To type the script by computer keyboard is much easier but to just click is the easiest action to do that is why we provide this facility to the user via Send SMS form as shown in Fig 71. This form also support the keyboard input to edit the script by the user if they want, however, as far as we provide this feature, no one want to type the script by them self, just use mouse, we can do nearly everything here.

As you can see, the name of each text box and check box of the first two frames are taken from **Output Port Setting** frame.



Analog Output port's Value

MOTOR01	A0	MOTOR02	CC

Digital Output port's Value

TV01	✓ ON	LAMP01	☐ OFF
TV02	☐ OFF	LAMP02	☐ OFF
DOOR01	✓ ON	LAMP03	☐ OFF
DOOR02	☐ OFF	LAMP04	☐ OFF

Keywords

☐ RESET	✓ REPORT
✓ RESET ALL	☐ READ

SMS Script for Output Board

```
<RESET
ALL><MOTOR01=A0><MOTOR02=C
C><TV01=1><DOOR01=1><REPOR
T>
```

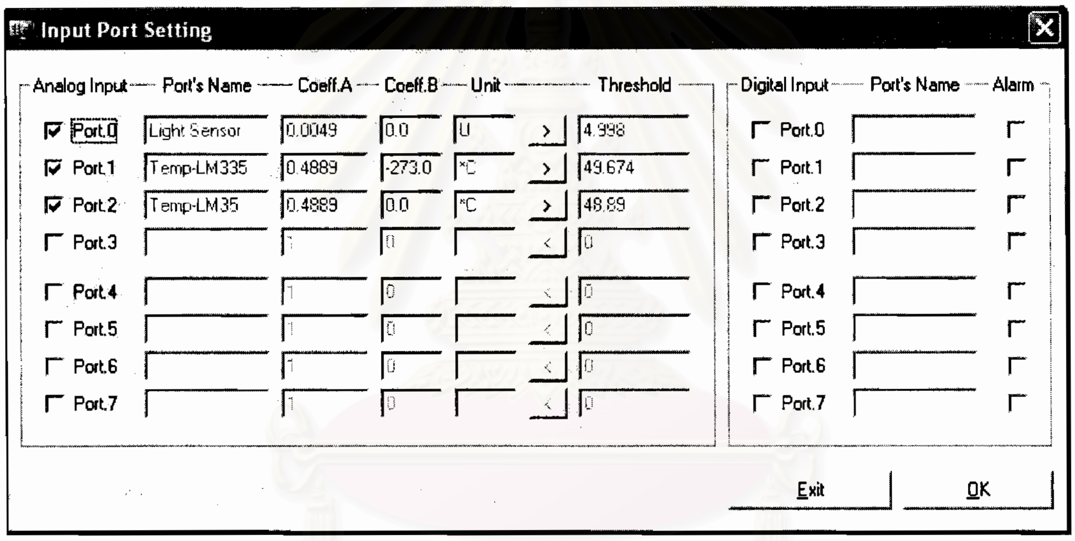
Fig 71. Send SMS Form

After selecting all the output port you want to activate/deactivate, just move the mouse to click on **Generate Script** button, the script will be send to the SMS text box automatically as shown in Fig 71. After your verification, you can send this message to a corresponding mobile node by just clicking on **Send** button.

Note that the script is executed line by line, word by word, so the first script will be taken in action first and then the next. By this way, some keywords have to be fixed when writing in to SMS text box. For example RESET ALL and REPORT. If we write RESET ALL after any script but before REPORT, we can say that the report message will be report all zero because of the action of RESET ALL or if we write REPORT first and RESET ALL at the last, the report will have no meaning because we don't know the current status of each output port after mobile node execute a script. However, **Generate Script** button will arrange this problem for you, but if you really want it to work different way as we do, you can edit the script by using the computer keyboard. One more thing, all the output value settled by this program will remain in the memory, so next time, if the port already activated/deactivated, and you want to send SMS again, those port will be displayed in blue color as show in Fig 71 to tell the user that, if everything is ok, the current values of the mobile node output port are the same as we display here.

Input Port Setting form: This form allows the user to enable or disable the input port of the mobile node. If the port is enable, the mobile node will scan/read the data from that port and store it in the memory (EEPROM) otherwise, the mobile node will skip that port in order to save the power consumption and memory as mention in Chapter 2.

To enable/activate a port, the user just checks on the corresponding Analog Input Port.X or Digital Input Port.X, where X is the port index from 0 to 7 as shown in Fig 72. From this figure, we can say there are only 3 analog port are activated, they are Port.0, Port.1 and Port.2. With this form, the user can give each port a virtual name according to the application. In here, Analog Input Port.0 is named as Light Sensor, Port.1 as Temp-LM335, etc.


Input Port Setting

Analog Input	Port's Name	Coeff.A	Coeff.B	Unit		Threshold
☑ Port.0	Light Sensor	0.0049	0.0	U	>	4.998
☑ Port.1	Temp-LM335	0.4889	-273.0	°C	>	49.674
☑ Port.2	Temp-LM35	0.4889	0.0	°C	>	48.89
☐ Port.3		1	0		<	0
☐ Port.4		1	0		<	0
☐ Port.5		1	0		<	0
☐ Port.6		1	0		<	0
☐ Port.7		1	0		<	0

Digital Input	Port's Name	Alarm
☐ Port.0		☐
☐ Port.1		☐
☐ Port.2		☐
☐ Port.3		☐
☐ Port.4		☐
☐ Port.5		☐
☐ Port.6		☐
☐ Port.7		☐

Fig 72. Input Port Setting Form

As the result read from any analog input port are stored in 10-bit binary format which correspond to 5V, we need to transform this dimension (binary value that correspond to 5V or voltage) to other dimension (°C or Kelvin) as we want by using the formula:

$$Y=AX+B$$

Where: A & B are the coefficient. X is original dimension and Y is final dimension. Take Port.1 as an example.

10-bit binary can represent up to 1023 unit that corresponds to 5VDC. However, from LM335, 10 mV corresponds to 10°C and at 20°C, the output voltage from LM335 is about 2.93 V, hence the coefficient A and B (CoeffA, CoeffB) are calculated as show below:

$$1023 \Leftrightarrow 5V$$

$$10mV \Leftrightarrow 10\ ^{\circ}C$$

$$\Rightarrow 1 \Leftrightarrow \frac{5V}{1023} \Leftrightarrow \frac{5V*100}{1023} = 0.4889\ ^{\circ}C$$

Hence, CoeffA = 0.4889

As LM335 give a reference temperature in Kelvin (at 0°C, we can find the output voltage from LM335 about 2.73V), so CoeffB = -273. Let verify these coefficients with following information: 2.93 V correspond to 20°C.

$$5V \Leftrightarrow 1023$$

$$\Rightarrow 2.93V \Leftrightarrow 599.478 \Rightarrow \text{take } 599$$

$$Y = 0.4889*599 - 273 = 292.8511 - 273 = 19.8511^{\circ}C \cong 20^{\circ}C \Rightarrow \text{OK}$$

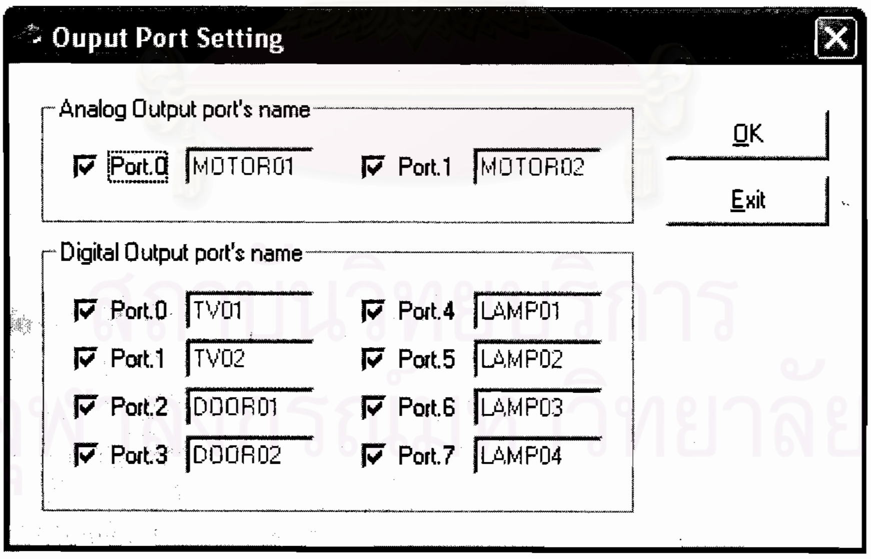
So with this linear transformation, we can transform a value from one dimension to a value in other dimension without any problem. However, as for photo resistor, the variation of its resistance versus light intensity is exponential function, not linear, which implies that this program is still limited but enough to survey or monitor its variation. If the user thinks that this is not enough, we can update this feature later by using the table, at this time, the user have to input the correct table value for the transformation function that make the non-scientific user get headache. For this reason, we stop our development here.

For the threshold value, the user can input any value in Threshold text box of the corresponding input port name. After this text box losses focus that means the user finishes input the value for this variable, so we can start our calculation transparently and change back this value to a correct value that can be used for mobile node because the mobile node have a limit resolution. From the value inputting by the user, we will convert it to a corresponding 10-bit binary and round it to 8-bit binary value then convert it back to original dimension. Because of the binary representation, that is why the correct value maybe differs from the original one that defined by the user. See Chapter 2 for the reason why we need to round this value to just 8-bit.

The comparison mode can be chosen by clicking on button > or <. This button will trigger automatically from one mode to other mode. After setting the input pin for mobile node, the user just click on **OK** button, we will record these setting, convert it to binary value and put it to a correct location in the system setting string, which is ready to write to EEPROM of mobile node.

Everything will be done automatically and transparently by our program to make user feel comfortable and have no headache while using this program.

Output Port Setting form: This form allows the user to define a virtual name for the output port name in order to be easy to remember. Each output name has a limit length to just 7 characters. This limitation is chosen, not because of our limitation but because we want to write many command in a limited SMS length. The number of enable port does not interfere the scanning process of the mobile node/central node, that is why we recommend that enable all the output ports and give them a name so you can easily control them via SMS anytime you want.



Analog Output port's name

☑ Port.0	MOTOR01	☑ Port.1	MOTOR02

Digital Output port's name

☑ Port.0	TV01	☑ Port.4	LAMP01
☑ Port.1	TV02	☑ Port.5	LAMP02
☑ Port.2	DOOR01	☑ Port.6	LAMP03
☑ Port.3	DOOR02	☑ Port.7	LAMP04

Fig 73. Output Port Setting Form

CHAPTER IV

PERFORMANCE EVALUATION

After everything has been implemented, we can now start to test our system performance by creating some scenario as describe below. In this chapter, we divide it into 3 sections, each section correspond to a scenario and in each section, the test condition and result will be described, analyzed and shown in order to prove/evaluate our system performance.

4.1 Data Collection, DTE-DCE Exchange Protocol and Error Detection

For this scenario, we let the mobile node to collect the data from 3 sources: luminosity (photo-resistor), temperature in Kelvin (LM335) and temperature in °C (LM35). These sensors are placed in Center of Excellence in Telecommunication Technology Laboratory (on my desk) to sense the luminosity and temperature in this room for a period of 6 days.

The settings for this system are:

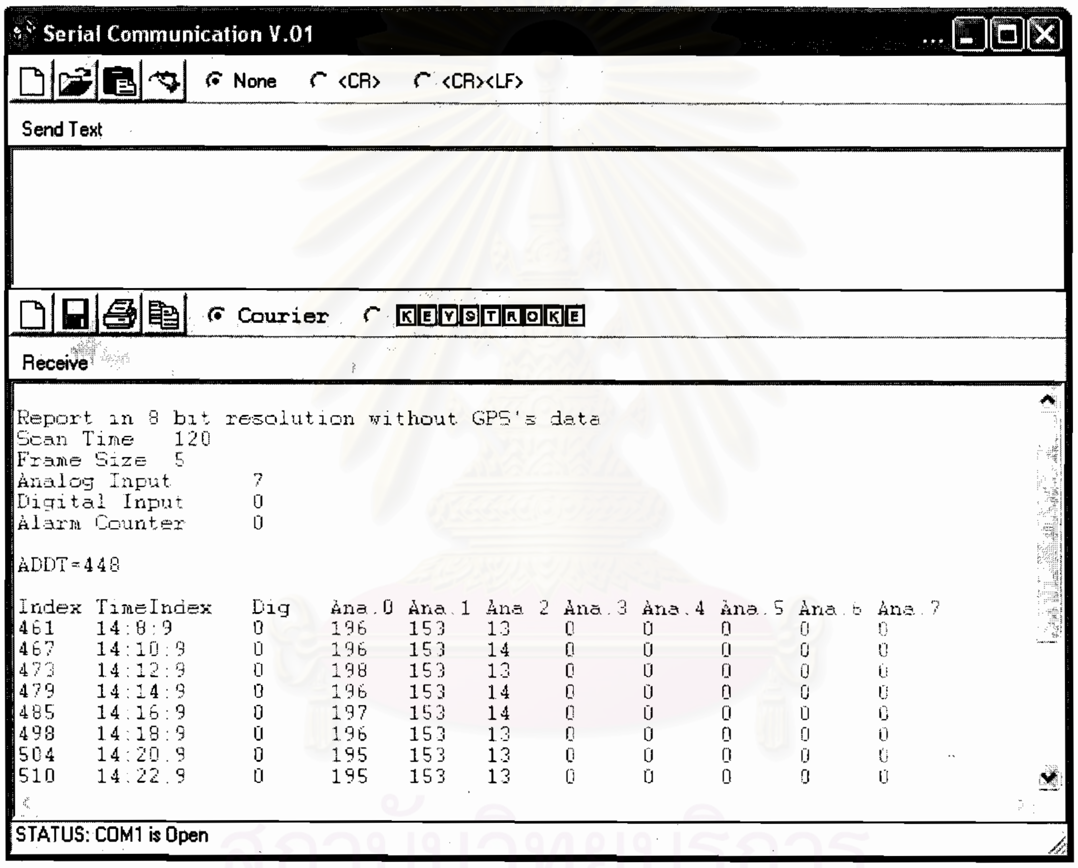
- Data Setting: See Fig 75
    - Sampling Period : 2 minutes
    - Frame Size : 5 samples/frame
    - Memory Size : 64 Kbytes
- Input Port Setting: See Fig 81
    - Port.0 Light Sensor : A=0.0049, B=0.0, Y=A.X+B > 5.00 V (4,985 volt)
    - Port.1 Temp-LM335 : A=0.4889, B=-273.0, Y=A.X+B > 49.67 °C (3,226 volt)
    - Port.2 Temp-LM35 : A=0.4889, B=0.0, Y=A.X+B > 48.89 °C (0,489 volt)

From this setting, we find that

- Total Mobile Node supported by this system is 1772 nodes
- Scanning Period = 6 days, 3.67H

We start running our system from Saturday, 05/03/2005 at 14:06:00 and stop it on Monday, 14/03/2005 at 16:21:38.

For this experiment, only one node is used. To verify the collected data, one computer is used to run our Modify HyperTerminal program to store the data sent by CPU module from debug pin. These data are reported in 8-bit format (not 10-bit format) because we just want to monitor whether the CPU module work well or not. If we use 10-bit format, we need to write the conversion function to change from 10-bi binary to printable ASCII character, which is not our target. For 8-bit, we can send it out directly with printable ASCII from 0 to 255 because this format is supported by USART as shown in Fig 74.



```
Report in 8 bit resolution without GPS's data
Scan Time    120
Frame Size   5
Analog Input       7
Digital Input      0
Alarm Counter      0

ADDT=448

Index TimeIndex    Dig    Ana.0  Ana.1  Ana 2  Ana.3  Ana.4  Ana.5  Ana.6  Ana.7
461   14:8:9       0      196    153    13     0      0      0      0      0
467   14:10:9      0      196    153    14     0      0      0      0      0
473   14:12:9      0      198    153    13     0      0      0      0      0
479   14:14:9      0      196    153    14     0      0      0      0      0
485   14:16:9      0      197    153    14     0      0      0      0      0
498   14:18:9      0      196    153    13     0      0      0      0      0
504   14:20.9      0      195    153    13     0      0      0      0      0
510   14:22.9      0      195    153    13     0      0      0      0      0
```

STATUS: COM1 is Open

Fig 74. Report Format Generated by Debug Pin of CPU Module, Received with our Modify HyperTerminal

From the debug-report in Fig 74, we can verify that:

- Scanning Time = 120 : means 120 s ⇔ 2 minutes
- Frame Size = 5 : means 5 samples/frame
- Analog Input = 7 : 7= %00000111 in binary, so it means that only Ana.0, Ana.1 and Ana.2 are enable.
- Digital Input = 0 : All digital inputs are disabled

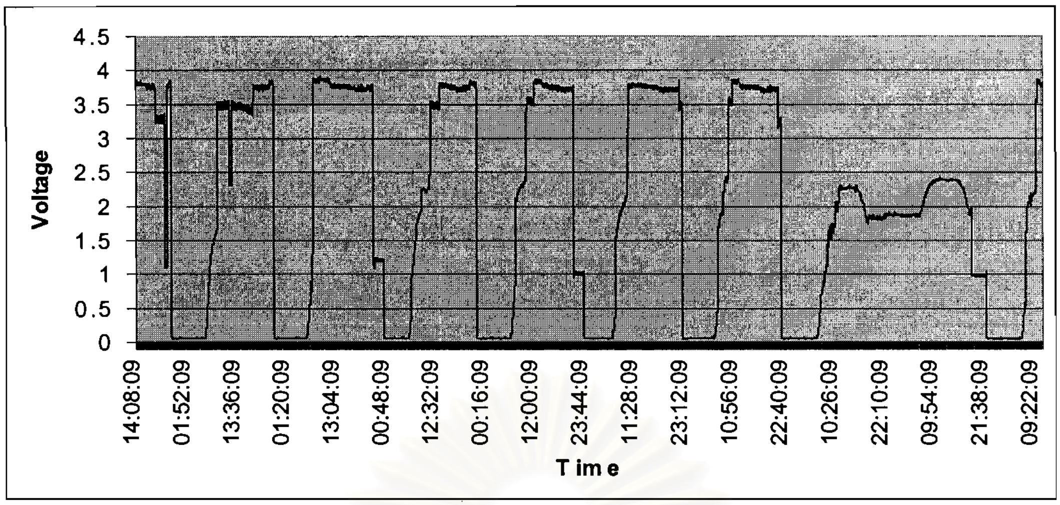


Fig 75. Variation of Light Intensity versus Time (8-bit)
Saturday, 05/03/2005 to Monday, 14/03/2005

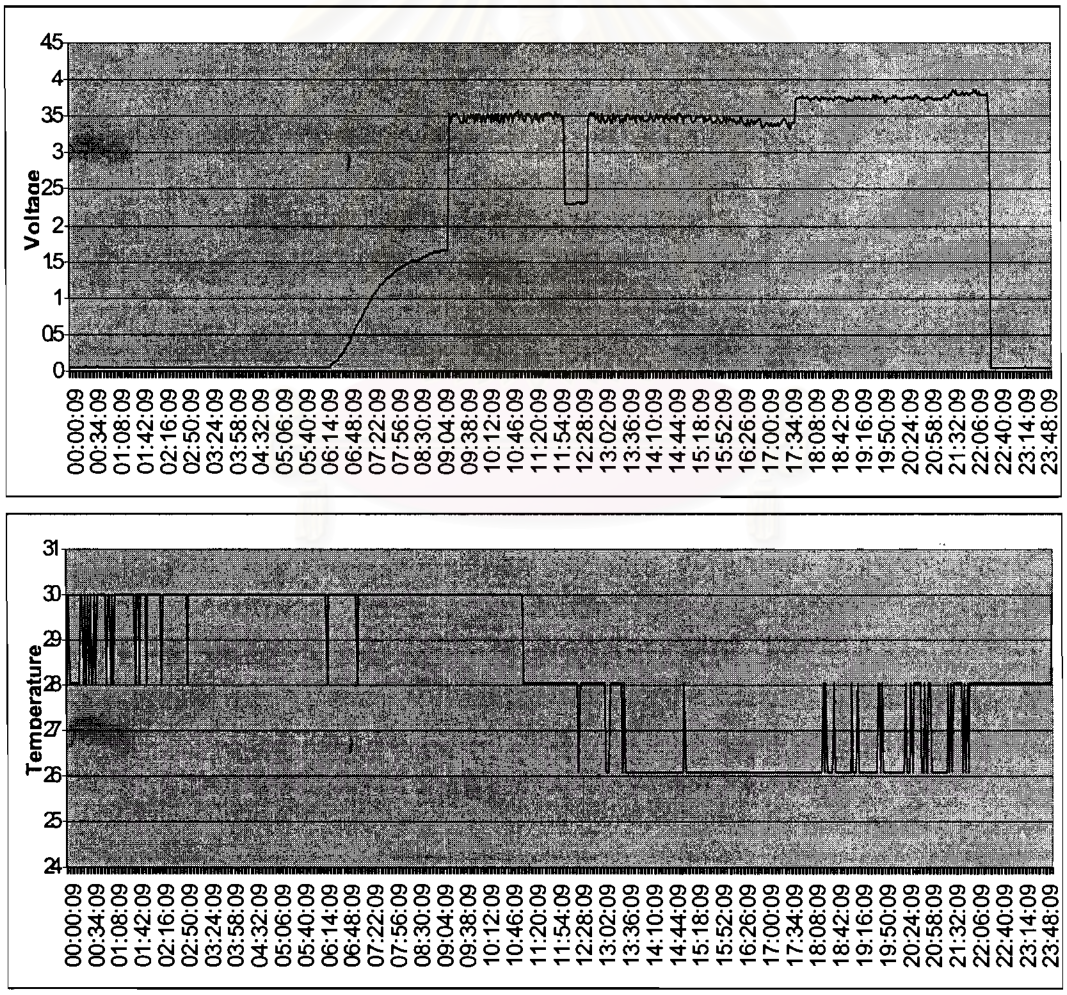


Fig 76. Variation of Light Intensity and Temperature (8-bit)
On Sunday, 06/03/2005

In order to be able to compare with the real data in the EEPROM, we open it with Microsoft Excel and write the formula to transform 8-bit binary to 10-bit binary and then use CoeffA and CoeffB to transform to corresponding dimension (voltage, degree C or Kelvin). The result of measuring light intensity by photo resistor after transformation is shown in Fig 75. Fig 76 shows the variation of light intensity and temperature on Sunday, 06/03/2005.

Now, let us interpret this graph before comparing it with the data in EEPROM to see what we can know from this graph/report.

Fig 75 illustrates that the light intensity is drop to minimum at night time and start to increase exponentially to the sunlight intensity (from 06:00 to 09:00, Fig 76 shows this variation more clearly) then about 09:00, the light intensity jump to high level and vary upon the turn-on/off of the neon lamp in the laboratory. According to the topology of this laboratory and the place where we install our sensor node, only me and other student sitting opposite to me that has more influence to this light variation than other students because when we come to the laboratory, we turn-on the light hence cause a big variation to the photo resistor. Other students also interfere to this value but not too much because they stay far away from our desk (from the sensor node). On Saturday night (12/03/2005), we can see that the light intensity did not drop down to minimum because on that day, some student come to the laboratory and forgot to turn-off the light when they went back home.

As mentioned in Chapter 3 (Section 3, Input Port Setting form), the variation of resistance of the photo resistor versus light intensity is not linear, imply the output voltage from resistor divider is not linear as well. One more, we do not have light meter (photographic equipment that measures the intensity of light) to compare with the output voltage of resistor divider in order to find the transformation coefficients (CoeffA and CoeffB), therefore, we leave it in voltage dimension not in lux or candela (1 lumen per square meter = 0.0929 foot candle).

Fig 76 shows the variation of light and temperature on Sunday, 06/03/2005. From 00:00 to 06:00, the light intensity in the laboratory is still at the minimum level. In this period, the temperature in this room increase little by little as the daytime is coming. From 06:00 to 09:00, the light intensity increases with the presence of sunshine in the room. However, as the room is not open to the sunshine (because of the curtain), the light intensity is limited (in voltage, the maximum of this variation is about 2.3V) comparing with the presence of neon lamp while it is turned-on as we can see a big gape from 1.7V to 3.5V at 09:00. From 09:00 to 17:30, the light intensity seems to be stable around 3.5V then jumps to 3.8V because at that time, other student came and turn-on other lamp. We can see that around 12:00, the light intensity dropped down to 2.3V because at that time, we went out to have lunch and then turn off the light, so this value came from the sunlight, not from neon lamp. We also can see that the light intensity from lamp source is noisier then from sunlight. This is due to the random process of photon created in the neon lamp (Poisson law). About 22:00, every one went out and all the lamps are switched off, therefore the light intensity dropped to the minimum immediately. For the temperature in this room, at 09:00, we turned on the air conditioner, so the temperature decreased little by little from 30°C to 26°C and then increased little by little at 15'clock.

Fig 76 also shows that the resolution of the temperature is about 2°C. This is due to the resolution of DAC. After round the data to 8-bit, this ADC is the same as 8-bit ADC. From section 3.3, we can see that:

$$\begin{array}{llcl} \text{10-bit} : & 1023 & \Leftrightarrow & 500°\text{C} \\ \Rightarrow \quad : & 1 & \Leftrightarrow & 0.4887°\text{C} \cong 0.5°\text{C} \\ \\ \text{8-bit} : & 255 & \Leftrightarrow & 500°\text{C} \\ \Rightarrow \quad : & 1 & \Leftrightarrow & 1.96°\text{C} \cong 2°\text{C} \end{array}$$

Now, look at the result reported by the CPU via Radio Link module. Fig 77 shows the variation of the light intensity like Fig 75 and Fig 78 show the light intensity and temperature variation like Fig 76. However, Fig 77 and Fig 78 are obtained via wireless connection, not direct from CPU like Fig 75 and Fig 76.

We can see that Fig 77 shows the same thing as Fig 75, it means that the communication between the central node and mobile node via air modem is established so well, no interruption or loss of data. This result can prove that our exchange algorithm and other associated algorithm, implementing in central node and mobile node, for the transmission work as expected.

The same for Fig 78 and Fig 76, however, Fig 78 shows the temperature variation more smoothly because the resolution of our ADC is 10-bit that implies 0.5°C resolution. From this figure, the minimum temperature is about 25.5°C (at 14'clock) and the maximum temperature

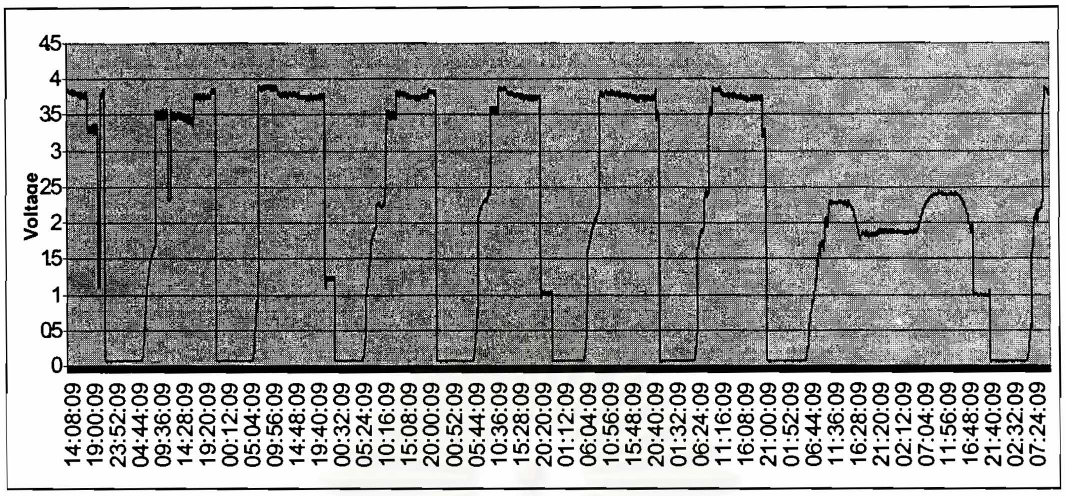


Fig 77. Variation of Light Intensity versus Time (10-bit)
Saturday, 05/03/2005 to Monday, 14/03/2005

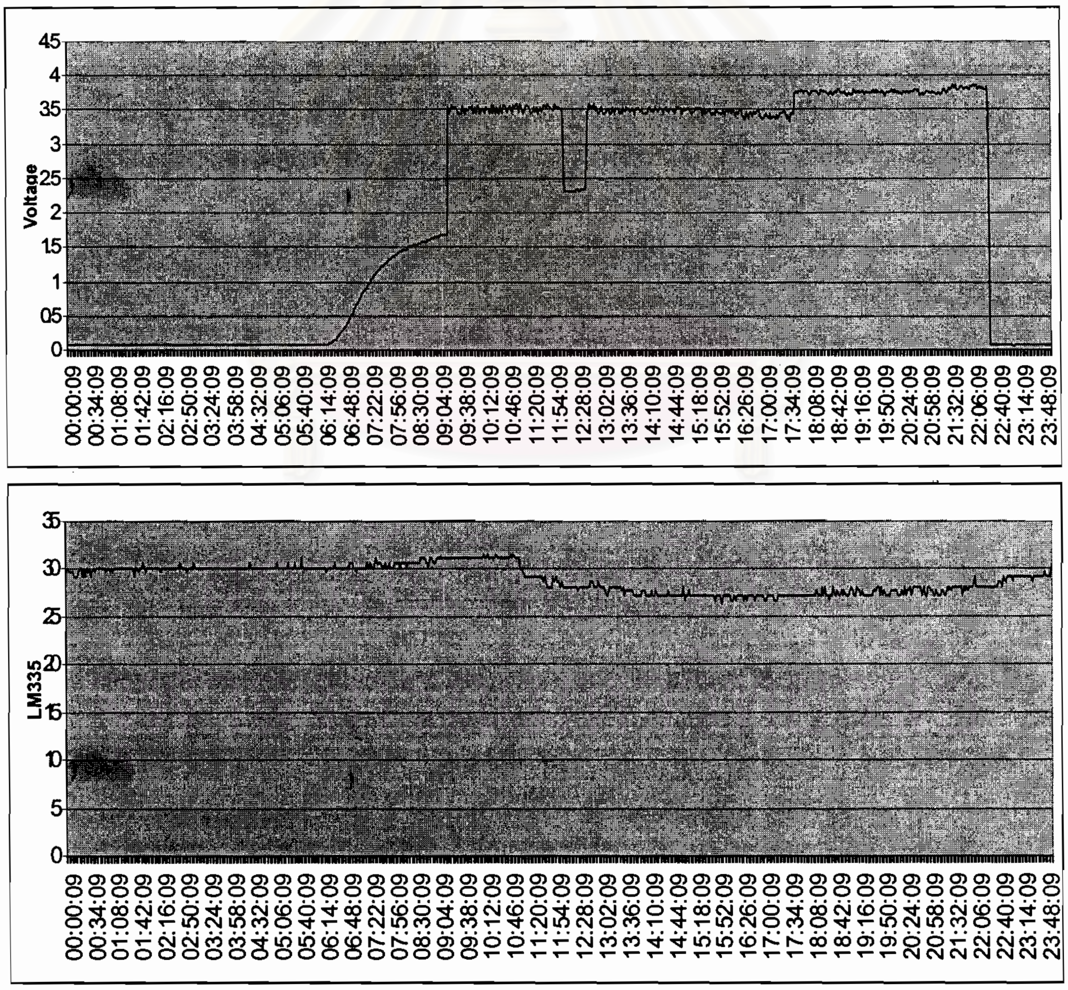


Fig 78. Variation of Light Intensity and Temperature (10-bit)
On Sunday, 06/03/2005

is about 31°C (at 09:00). The temperature increased little by little from 00:00 to 09:00, then dropped around 11:00 and reached the minimum around 14:00 then went up little by little from 20:00.

Just these two experiments, we can test out exchange algorithm whether it works well or not. Moreover, before we transfer these data via wireless link, a copy version of this data is made via a PC in order to verify it with 10-bit to 10-bit resolution. As the result shows that a copy from the memory and a received version from wireless link are the same.

Fig 79 shows the comparison result of LM335 and LM35. As mentioned in section 3.1.10, LM335 needs calibration that is why we cannot get the same result (we calibrate by hand); however, the results from LM335 can be accepted because it is just 1.77°C. We can adjust this difference by changing CoeffB from -273 to -274.77. Other remark, the result from LM35 look noisier then LM335 because LM335 has low pass filter at its output, where as LM35 just has a heavy capacitive load protective. Anyways, these results are applicable for monitoring application because the error is just 0.5°C. An alternative solution to improve these results is to use active low pass filter (amplifier + low pass filter) to improve both problems (noise and heavy capacitive load).

However, we do not satisfy at this point, we would like to test our program flexibility. To do this, we change the copy version of this data, save it back to the mobile node and then retrieve it via mobile phone. What we have changed are:

- Delete Frame header to delete the frame synchronize information. This information is very useful as we mention in chapter 2. For this modification, we first delete the complete 7 bits of this frame header at location 5776 (14:06:09), then delete just 5 (odd number) at location 5998 (15: 06:09), then we delete 6 (even number) at location 6220 (16: 06:09), and finally, we delete totally one frame at location 6442 (17: 06:09).
- At location 6664 (18: 06:09), we delete the data in the frame just 1 byte, then 2 bytes at location 6886 (19: 06:09).

The result of this modify version are shown in Fig 80. From this figure, we can see that there is no data loss at 14:06:09 because we lost only frame header not data. This is done by our program. It generates automatically time information for that data (data in the frame that we delete its header) from data of old frame information. At other points

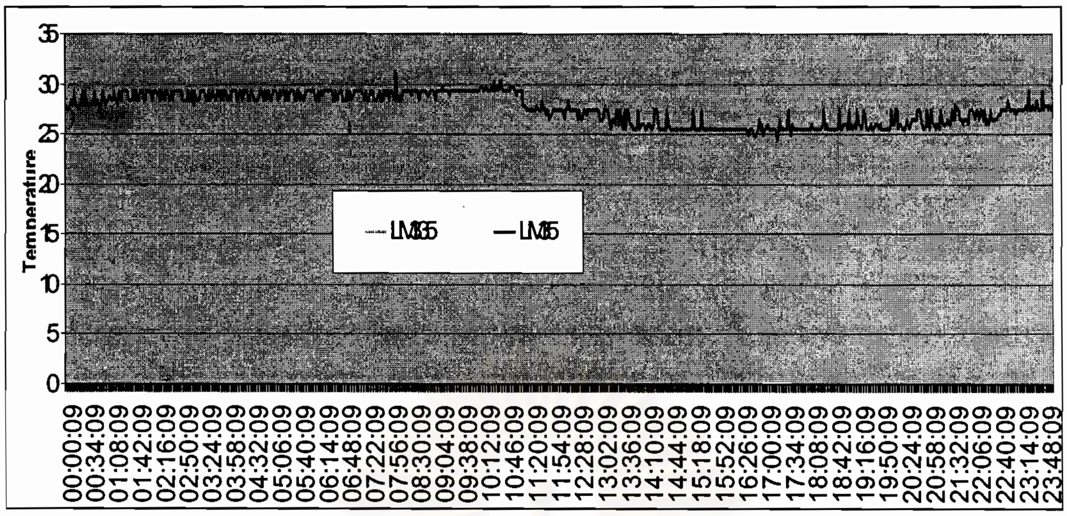


Fig 79. Comparison between LM335 and LM35

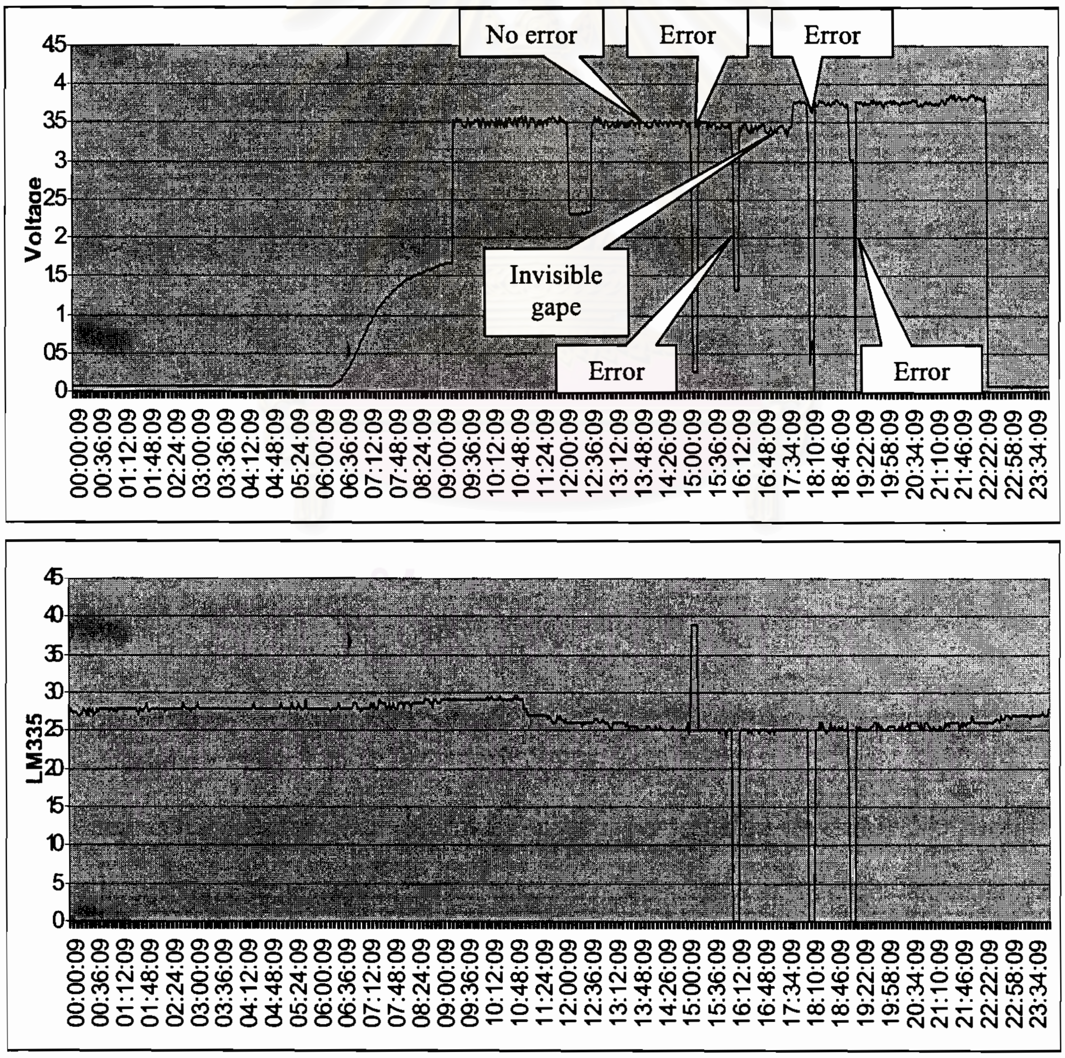


Fig 80. Variation of Light Intensity and Temperature (10-bit with errors) on Sunday, 06/03/2005

(15:06:09, 16:06:09, 18:06:09 and 19:06:09); the data are interrupted because we read it at wrong place. However, the data are interrupted only in that frame, when the next frame header is detected; their following data will be recovered normally.

So we can see that event the error propagates from one sample to other sample, this error cannot propagate to corrupt other frame, the interrupted data is just only in that frame. For the point 17:06:09, the data is lost completely, no time information, and represented as a gape, but only in this frame, the next frame will be recovered normally. Note that even the data is lost more then one frame; we only put it to 0 only for one frame.

So, if we set the frame size too big, and if such error happened it could infect many information. However, if we set the frame size too small, we lost the memory to store the data because one input take only 2 bytes for analog input and 8 digital input take only 2 bytes but only 1 frame header take the memory up to 7 bytes. So it is you to decide this value. Fortunately, if the error happened at the end of frame, the error data propagates only from that point to the end of that frame and if it was lost, we will fill it with 0 and this 0 will propagates in the same way.

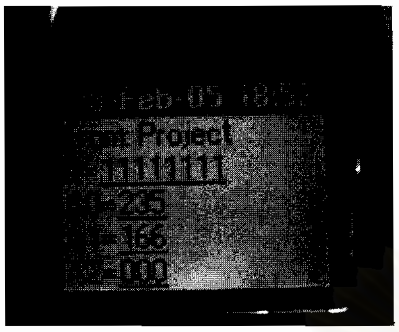
4.2 Alarm by SMS and Script Execution

To test this feature, we just keep the system setting as the first scenario, connect Debug pin of Radio Link module to the computer running our modify HyperTerminal and then test its functionality one by one as shown below:

<READ> and <REPORT> command: After sending one of these script commands to the mobile node, the mobile node will report a SMS as shown in Fig 81 and Fig 82.

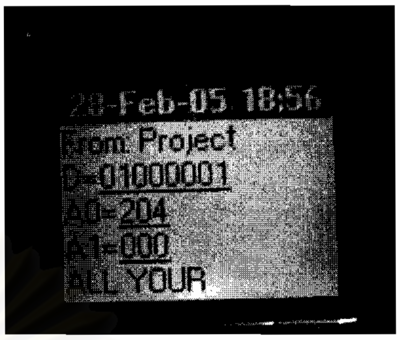
For <RESET ALL>, <RESET> and other script for output port can be verify by PC via debug pin. The internal process is shown in Fig 83. The script that we send look like Fig 84 and the corresponding reply message are shown in the preview Fig 81 and 91.

One more, we create undesired input for the CPU to see whether it can send us the critical message or not. As the result, we get a message as shown in Fig 85. Finally, we just let this mobile node to store the data until the address counter (AddT) surpasses $CCCC to see whether the mobile node can send us the buffer overrun message or not. As the result, we get a message as shown in Fig 86.



```
-Feb-05 18:5
om: Project
11111111
A0=235
A1=166
=000
```

Fig 81. <READ> Message

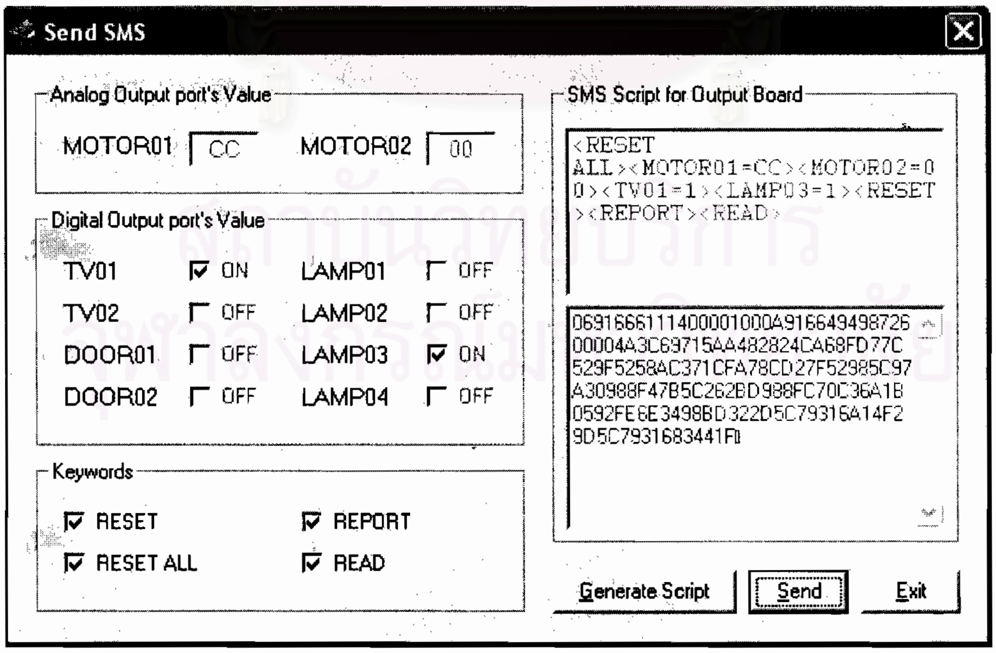


```
28-Feb-05 18:56
From: Project
D=01000001
A0=204
A1=000
ALL YOUR
```

Fig 82. <REPORT> Message

Internal Process		Explications
RSA		<RESET ALL> is executed
A0=204		Analog Port.0 is set to 204 = CCh
A1=0		Analog Port.1 is set to 0
D0=1		Digital Port.0 is turn on
D6=1	⇔	Digital Port.6 is turn on
RS		<RESET> is executed
RP		<REPORT> is executed
RD		<READ> is executed

Fig 83. Radio Link Internal Process



Send SMS

Analog Output port's Value

MOTOR01: CC   MOTOR02: 00

Digital Output port's Value

TV01 [✓ ON]   LAMP01 OFF
TV02 OFF   LAMP02 OFF
DOOR01 OFF   LAMP03 [✓ ON]
DOOR02 OFF   LAMP04 OFF

Keywords

[✓ RESET]   [✓ REPORT]
[✓ RESET ALL]   [✓ READ]

SMS Script for Output Board

```
<RESET
ALL><MOTOR01=CC><MOTOR02=0
0><TV01=1><LAMP03=1><RESET
><REPORT><READ>
```

```
06916661114000010000A91664949872б
00004A3C69715AA482824CA69FD77C
529F5258AC371CFA78CD27F52985C97
A30988F47B5C262BD988FC70C36A1B
0592FE6E3498BD322D5C79316A14F2
9D5C7931683441F0
```

Fig 84. Script Command Sent from Our Program

From these figures, we can verify that our Read SMS, Script execution, Send SMS and other association functions that we implemented in Radio Link and CPU module work as expected.

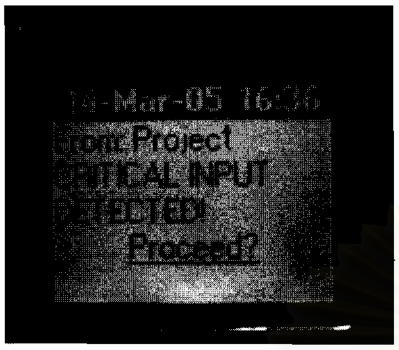

14-Mar-05 16:36
rom: Project
RITICAL INPUT
ETECTED!
Proceed?

Fig 85. Critical Input Message

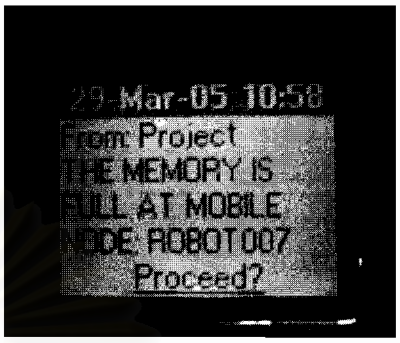

29-Mar-05 10:58
rom: Project
HE MEMORY IS
ULL AT MOBILE
ODE: ROBOT 007
Proceed?

Fig 86. Buffer Overrun Message

It seems that our chapters 2 and 3 is too long and have many things to describe but in this chapter, we just have little thing to analyze. This is not strange; a hard beginning makes a good ending. Fig 87 shows the perspective of our mobile node.

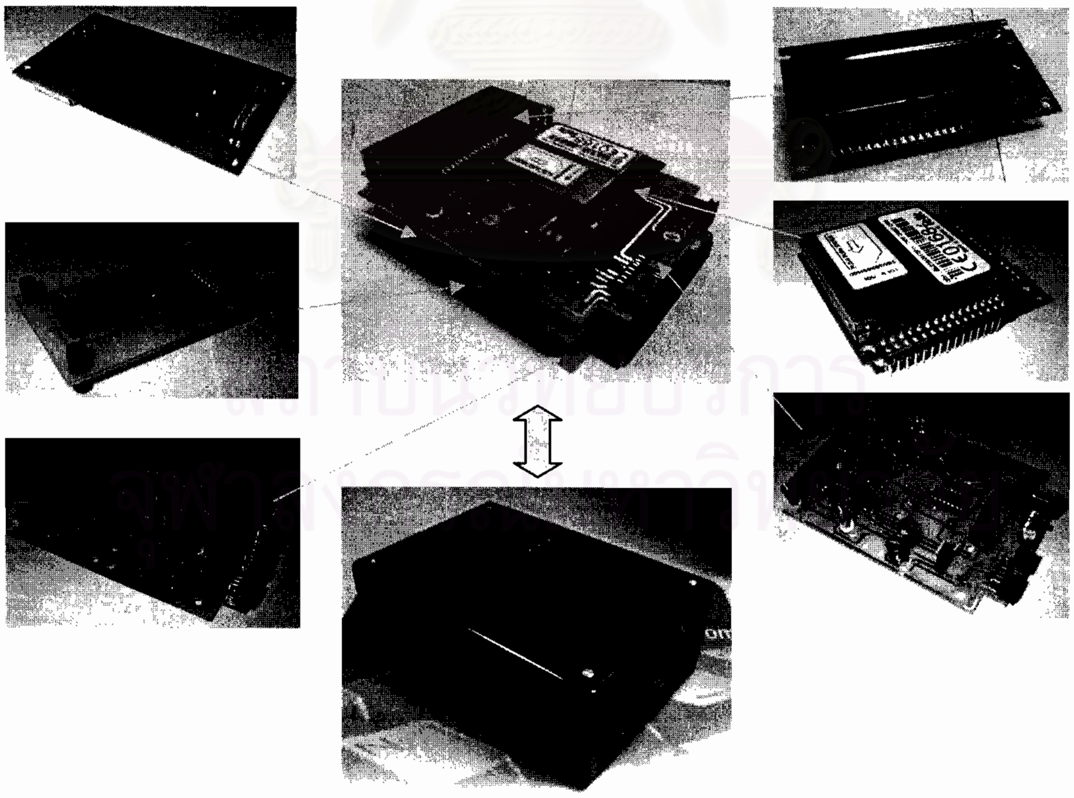


Fig 87. Perspective View of Mobile Node

CHAPTER V

CONCLUSION AND RECOMMENDATION

This research tries to implement and evaluate the system performance of mobile node and central node for Tele-measurement Sensor Network that are based on hardware and software implementation.

We have described the algorithms and the reason why these algorithms are chosen. We have implemented the mobile node, sensor board, some output boards, programming and debugging board and written programs in VB in order to test the system performance as well as to make a complete system that works in reality, not only simulation environment.

We evaluate the system performance by running our program (Central of Tele-measurement Sensor Network) in a window platform and turning on our mobile node to collect the data in real environment, then retrieving that data and comparing with the data that we get from other program (modify HyperTerminal). We also add some errors to that data in order to test the intelligent capability of our program.

The result has shown that all the features/functionalities work as expected (both sides) and we note that the frame size plays a very important role for error detection feature and data protection from propagation error. However, the frame size also influences the scanning time since it needs more bytes to store its information (frame header).

We also observed that the low pass filter can help reducing the noise (electromagnetic and Poison noise, etc.) to get smoother result. Poor quality power-supply also causes a noisy result since it supplies unstable voltage to the ADC module in PIC.

By using the mature technologies (GSM, Microcontroller, etc.), we can guarantee the stability, reliability and durability of our system.

By using the modular technique, we can update our system to feed the new environment/system/need easily.

FUTURE WORK

This system can work properly in the area where it is covered by the GSM network. The extended version of this mobile node (combined with Ad-Hoc network technology) should be implemented in the future research in order to extend the coverage area of the system.

This system has only one central node to control the mobile nodes and to store/update all the data retrieved from mobile nodes. The extended version of this central node (combined with internet connection) should be implemented in the future research in order to be able to extend the coverage area as well as the control/remote node of the system.

REFERENCES

1. Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A Survey on Sensor Networks. Georgia Institute of Technology, IEEE Communications Magazine. (August 2002): 102-114.

2. David Shepherd. Networked Micro-sensors and The End of the World as We Know It. IEEE Technology and Society Magazine. (Spring 2003): 17-21.

3. Yoshihiro Kawahara, Masateru Minami, Hiroyuki Morikawa and Tomonori Aoyama. Design and Implementation of a Sensor Network Node for Ubiquitous Computing Environment. Proceeding of the 58th IEEE International Conference on Vehicular Technology. 5 (October 2003): 3005-3009.

4. Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Design and Implementation of Wireless Sensor Networks for Habitat Monitoring. Intel Research Laboratory, Berkeley Intel Corporation, EECS Department University of California at Berkeley, College of the Atlantic Bar Harbor. http://www.cs.berkeley.edu/~polastre/papers/masters.pdf, 2003.

5. Ara N. Knaian. A Wireless Sensor Network for Smart Roadbeds and Intelligent Transportation Systems. Massachusetts Institute of Technology. http://www.media.mit.edu/resenv/pubs/theses/AraKnaian-Thesis.pdf, 2000.

6. Traig Born and Joel Glidden. Computer Systems Senior Design: A Linux-Based Robot Control System. University of Arkansas at Little Rock, Donaghey College of Information Science and Systems Engineering. http://theduchy.ualr.edu/classes/syen4386/cs_sd.pdf, 2003.

7. http://www.microchip.com

8. http://www.atmel.com

9. http://www.dalsemi.com

10. http://www.phillip.com

11. Chuck Hellebuyck. Programming PIC Microcontrollers with PicBasic. Newnes, 2003.

12. Christian Tavernier. Les Microcontrôleurs PIC Applications. DUNOD, 2000.

13. PicBasic Pro Compile. Micro Engineering Labs, 1999.

14. John IOVINE. PIC Microcontroller Project Book. McGraw-Hill, 2000.

15. Scott B. Guthery, and Mary J. Cronin. Mobile Application Development with SMS and the SIM Toolkit. McGraw-Hill, 2002.

16. Evangelos Petroutsos. Mastering Visual Basic 6. SYBEX, 1998.

17. John Connell. Beginning Visual Basic 6 Database Programming. Wrox Press, 1998.

18. http://www.national.com

19. http://www.linear.com

List of Publications

Design and Development of Mobile and Central Nodes for Tele-measurement Sensor Network. ECTI Annual Conferences (ECTI-CON), 12-13 May 2005, Pattaya, Thailand.

Design and Development of Mobile and Central Nodes for Tele-measurement Sensor Network

KY-Leng and Watit Benjapolakul

Department of Electrical Engineering, Chulalongkorn University, Bangkok 10330, Thailand
E-mail: lengcallrobot@yahoo.fr, watit.b@eng.chula.ac.th

ABSTRACT

The conventional environmental network does not provide the feedback ability and is not interesting for the non-scientific user. So, we propose a personal sensor network for environmental monitoring and remote security system for company, organization and home user. This network is fast, easy to build and allow the end user to change the configurations and easy to update the system by using the modular technique and ability to setup new configuration from center. It can be used as indoor or outdoor network. By choosing an appropriate technology from Microchip and algorithm, we can optimize the power consumption. In this paper, we will describe in detail about hardware and software needed for developing this network (central and mobile nodes).

Keywords: Mobile Node, Tele-measurement, Sensor Network

1. INTRODUCTION

Recent advances in wireless communications and electronics have enabled the development of low cost, low-power, multifunctional sensor nodes that are small in size and can communicate undeterred in short distances. The sensor network represents a significant improvement over traditional sensor and it can be used in various application areas such as health, military, industry, home, etc. For different application areas, the sensor networks are designed with different technologies [1-2]. It is also a key technology to obtain user's context in the real world because it can enable long-term data collection at scales and resolution that are difficult, if possible, to obtain otherwise. A large number of researchers have been working on sensor network technologies primarily for the realization of large scale environmental monitoring systems for military use and/or scientific use [1-4] and in this recent year, a large number of systems have been proposed for a small network and indoor network such as MICA [3] and U^3 node that is capable of communicating with other nodes to carry sensing data and queries issued by users. Wireless sensor package developed by Ara N. Knaian can be used only for Intelligent Transportation Systems by counting passing vehicles, measuring the average roadway speed and detecting ice and water on the road [5]. However, they are not applicable for a non-scientific and home user and have no feedback path. A Linux-Based Robot Control System [6] developed by Traig Born and Joel Glidden is very powerful in term of data collection and remote control, however, it cannot be considered as a sensor network because of its power consumption and its size. The Personal Tele-measurement Network is a private network, easy to update and allows the users to change the configuration by themselves. It is an outdoor and indoor network that can be used in a large or small-scale network such as environmental monitoring and remote security system. It is very difficult to envision how such future application should be like, however it is desirable to specify or list some applications, so we try to show some real applications as listed below:

- Measure the water level, gas toxicity, air pollution rate, humidity, and luminosity and control the dam gate.
- Measure the current intensity of the power distribution sub center, alert the main center with the cause and cut off the dangerous line at the exact place.
- Use as security guard: monitor the gate, doors, windows, hallways, alert the security office and/or police stations with the location where the unpleasant event have been occurred.
- Control the electrical appliances in the house.

2. HARDWARE & SOFTWARE DESIGN FOR THE MOBILE NODE

2.1 Hardware:

In order to fulfill the flexible requirements, we divide the system into small modules (modular system), which have specific functionalities, as shown in Fig. 1.

Radio Link Module: Use a mobile phone kit as the radio link module to ensure the data transfer between central and mobile node (MN). We use one PIC16F876 (Programmable Integrated Circuits) and one EEPROM (Electrically Erasable Programmable Read Only Memory) to ensure this functionality.

CPU Module: use PIC16F877 to ensure the data processing, sampling and storing data in the external EEPROM. This module communicates with the Radio link module, some external sensor boards & Real Time Control (RTC) via I^2C protocol.

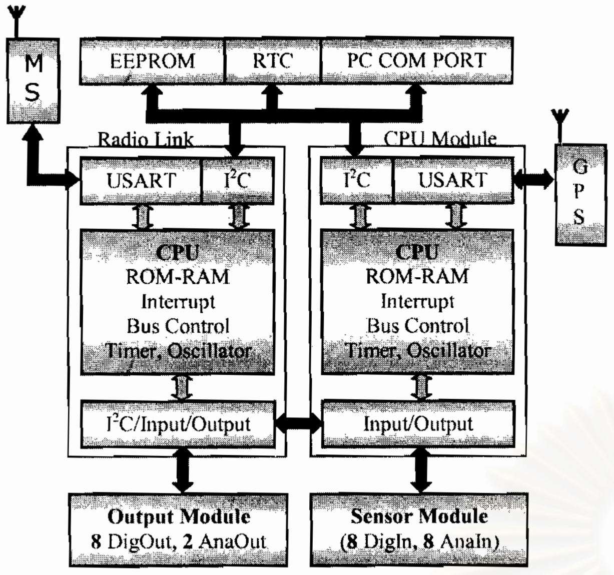


Fig. 1: Mobile Node Architecture.
MS: Mobile Station, GPS: Global Positioning System Receiver.

Sensor Module: As the CPU module supports digital & analog input, some sensors can connect directly to digital input port and other sensors that give the result as voltage (0V to 5V) can connect directly to analog input port. We also reserve one RS232 port for GPS receiver.

Output Module: We fix to use I^{2}C-Digital to Analog Converter AD5301 or PCF8591 to provide two analog outputs for the user with 8 bits resolution and PCF8574 to provide 8 digital outputs.

Note: All these features are supported by our CPU module. If we want to use it, just connect the corresponding module then enable it from PC (Personal Computer). This implies flexibility and saving money because we pay for what we want. The modular technique provides not only flexibility but also minimizing the power consumption as the CPU module can go to sleep mode (I<2 mA at 5V, 4 MHz) after processing all its works (it take about 300 ms/scan, so if we scan the input every 1 min ⇒ CPU can go to sleep mode for 59.7s ⇔ 99.5% of time). Because the GSM module & Radio Link module cannot go to sleep mode, we do not say that our system consume less power than other systems, however, we try to minimize it as much as we can.

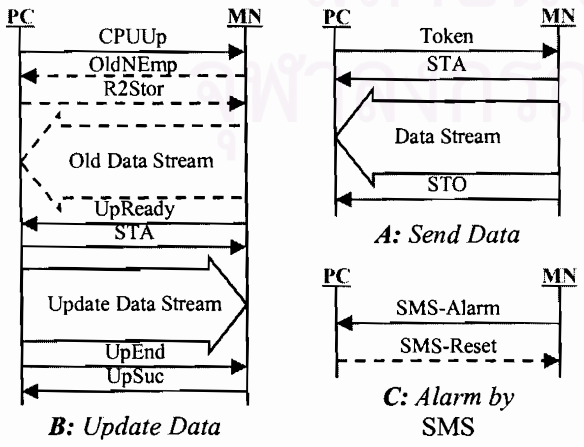


Fig. 2: Time diagram for the Exchange of Command and Response Signallings

***Table 1:** Protocol Constants*

Name	ASCII Character	Description
STA	178	Indicate the Start of data
STO	187	Indicate the End of data
Token	196	Request for Sending data
CPUUp	205	Request for Updating data
R2Stor	214	Ready to store the old data
UpEnd	223	Indicate the End of Update-Data
OldNEmp	232	Data left in EEPROM
UpReady	241	Ready to store Update-Data
UpSuc	250	Updating is finished successfully
S	167	Synchronizing Frame Header

2.2 Software:

We implement a layer-2 protocol to Mobile node in order to archive the flexibility to change the configuration by the user during or after the network configuration. We also provide the feedback control by implementing a script execution function.

Radio link module: We use AT-Command to send the data and SMS (Short Message Service) via Air-interface (mobile phone), however, as not every mobile phone supports text mode; we implement text-to-PDU mode algorithm [7] for we can support the majority of mobile phone. We also implement a layer-2 protocol on it in order to ensure data transfer between the central and mobile node as shown in Fig. 2. Send the collected data, update the user customization/setting, alarm warning, send input data, feedback control and send its report are the functionality supported by this module. All information about this protocol is shown in Table 1 with other constants used in our work.

Note: All the user customization, warning message and acquired data are stored in the external EEPROM, which is AT24C512 (512kbits ⇔ 64kbytes). This EEPROM stores, not only the data for *Radio link module* (central mobile phone number and manager mobile phone number), but also a very useful data for *CPU module* (options for CPU: scanning period, frame size, input masks, input critical values, digital output port's name, analog output port's name, upper bound counter limit and alarm flag). Here, the sampling period is the same for all input ports because we do not know exactly the future application of each port. After the network configuration, the user can only change the options for CPU. By just changing the data at the corresponding memory location, we can allow the user to customize the network very easily during installation as well as after installation.

CPU Module: Ensure the data processing function. We sample the input port at every sampling period

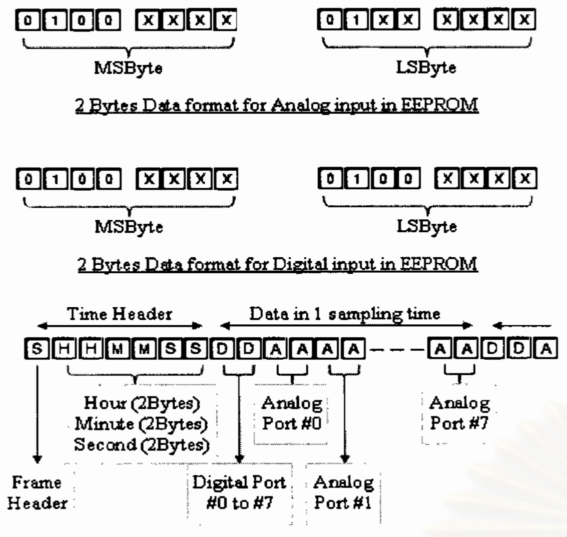


***Fig. 3:** Data Format in EEPROM*

(integer value from 1 to 60 second or minute) and store the data in the external EEPROM. Note that not all the inputs are sampled; we sample only the valid input ports defined by the input mask (*to reduce power consumption and storage*). Fig. 3 shows the data format for analog input, digital input and frame format that we use in our program. We suppose that the digital input and all the analog inputs are enabled.

The fact that we set bit #6 of each byte to 1 is to avoid the data to be special characters such as character 0, character 10, character 13, and character 26, etc. these characters are considered as a command, not data, for modem and cause modem to disconnect without sending disconnect command because the data themself have the same meaning as a command.

Sending data process is controlled by the central node, it looks like "one-to-many" relationship, only the center computer can ask and the other nodes can only answer. In case that the CPU module stores the data more than FFE0h, the CPU module will inform the Radio link module to send the SMS with the message "*The memory is full at mobile node:* **ID**" where **ID** is the mobile node itself. At the same time, the CPU continues to store the new data until it reaches the full memory then restart again. However, this is very rare because we already thought about this problem and we finally found the algorithm then program it at the center computer. See also Fig. 4.

In case that any input data reaches the critical value (data is smaller/bigger than threshold value, predefined by the user, for analog signal or data is different from user predefined value for digital signal), we count up the counter by 1. We continue to count up whenever any input is critical and the counter value is less than the upper bound counter limit. When all inputs change to a normal value, we count down the counter by 1 and continue to count down if all the next sampling inputs are

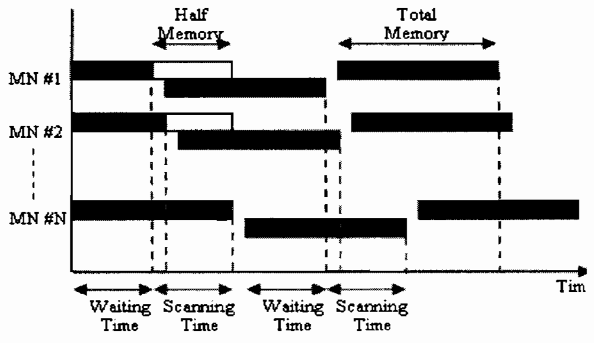


***Fig. 4:** Scanning Process Chronogram*

normal and the counter value is bigger than 0. Assume now that the counter reaches the upper bound counter limit, the CPU Module will tell the Radio link module to send alarm message via SMS with the current input value from all ports beginning by the message "*Critical input detected!*" and then reset the counter. After the center computer received the alarm message, it can send the feedback command via SMS (script format) to the mobile node to activate some output ports and reset alarm (it's up to the user customizations). Script format is defined by character 60 and character 62 in ASCII mode. Any text begins with ASCII character 60 and ends with character 62 will be considered as command word, for example: <RESET> is used to reset alarm. The reason why we implement this format is to avoid the accidental error when unknown user sends a wrong message to this mobile phone. In case that we do not send the reset alarm, the mobile node will continue to report the critical port whenever the counter reaches the Upper bound.

3. HARDWARE & SOFTWARE DESIGN FOR THE CENTER COMPUTER (CENTRAL NODE)

3.1 Hardware

We need one mobile phone connected to the COM port. Right now, we use ERICSSON T68. We implement a Serial-to-I^2C interface adapter for the Central node in order to provide the end-user the ability to customize their network during installation.

3.2 Software

We implement the same layer-2 protocol (as for Radio link Module at Mobile node) and **I^2C** protocol in the Central node by using the VB programming language (Visual Basic). Moreover, we create **G**raphic **U**ser **I**nterface and **database** (user Application layer) for storing the Mobile node information such as mobile node ID, phone number, location (province, district, section, and description), and critical value, etc and also provide the ability for the user to customize his own network by changing the Input mask, Telephone number, etc. In short, the user can change the data needed for Radio link module and CPU module during the network installation. We also provide some functionality to the user to save the input data automatically to the user-predefined-folder, plot the record data like in Excel, print the record, show

how many mobile nodes this system can support with the current configuration, etc. The latter is implemented to avoid the problem of not enough memory to store new data at mobile node. After reading the message, the program will automatically delete the message from the center computer's mobile phone preventing out of memory in the center computer's mobile phone.

Note: If we set Sampling Period to 0, it means that we do not want to save the input data but scan as fast as possible (every 20 ms), so the GPS feature is disabled. In fact, we can sample the input faster than this; however, as we want to save the power and 20 ms sampling period is fast enough, it was decided as a sampling period for our applications.

To determine how many mobile nodes this system can support and how long we can store the data before send it to PC, the following formulas are used.

$$N_F = M / ((F_S*2) + H) \quad (1)$$
$$T = N_F*T_S* F_S \quad (2)$$
$$N = T/T_0 \quad (3)$$

Where:

N_F	:	Number of Frames contained in the last half memory.
M	:	Half Memory size constant. It is 32768 bytes for 64kbytes memory.
F_S	:	Frame size defined by the user.
H	:	Time Header. It is used to indicate the beginning of new frame. H = 7 bytes (Fig. 3).
T	:	Time needed to fill all that half memory.
T_S	:	Scanning period defined by the user.
T_0	:	Scanning time needed for 1 node. It is equal to 4 min for 64kbytes memory.
N	:	Number of Mobile Nodes supported by this system.

The longest Scanning time must be less than or equal to the Waiting time. See Fig. 4 for more detail.

4. ACTUAL WORKS

As now we finish implementing our layer 2 protocols (Fig. 2A and Fig. 2B), communication link between mobile node and central node, algorithm that we use to avoid the out of memory at mobile node, send/receive SMS and script execution, so we have tested the following applications:

- **Remote Control via SMS**: Dial with receive/send SMS, script execution and digital output [8].
- **Read temperature via SMS**: Dial with receive/send SMS, script execution and temperature sensor.
- **Monitor the current in power transmission line**: Dial with analog input, exchange command and signaling, data format and scanning process.

In all of the tested applications above, we are satisfied with the results expected.

5. CONCLUSIONS AND FUTURE WORK

These applications are starting points for our future work. It looks simple and different from our real target. However, it can let us know and understand more about what we need and what we should do in the following steps. We intend to design and develop a pilot platform of central and mobile nodes for tele-measurement sensor network, which can support the functionalities stated above in the very near future.

6. ACKNOWLEDGEMENT

The authors wish to thank the AUN/SEED-Net's Collaborative Research Support Project (JFY2004) for supporting this work.

7. REFERENCES

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A Survey on Sensor Networks", Georgia Institute of Technology, *IEEE Communications Magazine*, August 2002, pp. 102-114.

[2] D. Shepherd, "Networked Micro-sensors and the End of the World as we know it", *IEEE Technology and Society Magazine*, Spring 2003, pp. 17-21.

[3] Y. Kawahara, M. Minami, H. Morikawa, and T. Aoyama, "Design and Implementation of a Sensor Network Node for Ubiquitous Computing Environment", *Proceeding of 58th IEEE International Conference on Vehicular Technology*, Vol. 5, 2003, pp. 3005–3009.

[4] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Design and Implementation of Wireless Sensor Networks for Habitat Monitoring", Intel Research Laboratory, Berkeley Intel Corporation, EECS Department, University of California at Berkeley, College of the Atlantic Bar Harbor, http://www.cs.berkeley.edu/~polastre/papers/masters.pdf, 2003.

[5] A. N. Knaian, "A Wireless Sensor Network for Smart Roadbeds and Intelligent Transportation Systems", Massachusetts Institute of Technology, http://www.media.mit.edu/resenv/pubs/theses/AraKnaian-Thesis.pdf, 2000.

[6] T. Born and J. Glidden, "Computer Systems Senior Design: A Linux-Based Robot Control System", University of Arkansas at Little Rock, Donaghey College of Information Science and Systems Engineering, http://theduchy.ualr.edu/classes/syen4386/cs_sd.pdf, 2003.

[7] S. B. Guthery, and M. J. Cronin, *Mobile Application Development with SMS and the SIM Toolkit*, McGraw-Hill, 2002.

[8] D. Rey, *Interfaces GSM, Montages pour téléphones portables*, Editions Techniques et Scientifiques Françaises, Paris, 2004.

Author Biography

KY-Leng was born in 1979 in Battambang province, Cambodia. In 2001, he received his Dip. Ing. in Electrical Engineering from Institute of Technology of Cambodia (ITC), Cambodia. From 2001, he has worked at ITC as a full time lecturer. His research interests are in the mobile communication system, electronics and computer programming.