[UCI]I804:24011-200000232654

2006년 08월

박사학위논문

16채널 스트레인 게이지를 위한 무선 센서네트워크 시스템

조선대학교 대학원

제어계측공학과

원 용 일

16채널 스트레인 게이지를 위한
무선 센서네트워크 시스템

Wireless Sensor Network System

for 16 Channels Strain Gauge

2006년 08월 25일

조선대학교 대학원

제어계측공학과

원 용 일

16채널 스트레인 게이지를 위한 무선 센서네트워크 시스템

지도교수 장 순 석

이 논문을 공학박사학위신청 논문으로 제출함

2006년 월

조선대학교 대학원

제어계측공학과

원 용 일

원용일의 박사학위 논문을 인준함

2006년 월 일

조선대학교 대학원

Contents

List of Figures

List of Tables

List of Lists

ABSTRACT

Wireless Sensor Network System For 16 Channel Strain Gauge

Won Yong-Il
Advisor: Prof. Soon-Suck Jarng
Dept. of Control & Instrumentation Eng.
Graduate School of Chosun University

This paper presents a wireless sensor network system in which weight measuring instrumentation process, data conversion process, and the series of signal controlling processes are seamlessly cooperative. 16 strain gauges are working as incoming sensors. Each output is amplified and filtered for proper analog signal processing. Several measuring instrumentation OP amps and general-purpose OP amps are used. 12 bits A/D converter transforms analog signal to digital bits. PIC microprocessor controls the operations of measuring strain gauges. RF RS232C module is adapted for wireless communication between the PIC microprocessor and TCP/IP hosting device. Service program is considered as seamless and stable monitoring operation tool over Internet. Database-based data handling system could bring firm basis for statistical data processing application. In order to overcome the time sequence problem among different device parts, a total communication system using handshaking method has been developed.

Chapter 1 Introduction

1.1 Background

A sensor network is a computer network of spatially distributed devices using sensors to monitor conditions (such as temperature, sound, vibration, pressure, motion or pollutants) at a variety of locations. Usually the devices are small and inexpensive, allowing them to be produced and deployed in large numbers; this constrains their resources in terms of energy, memory, computational speed and bandwidth[4].

Each device is equipped with a radio transceiver, a small microcontroller, and an energy source, most commonly a battery. The devices work off each other to deliver data to the computer which has been set up to monitor the information. Sensor networks involve three areas: sensing, communications, and computation (hardware, software, algorithms). They are applied in many areas, such as video surveillance, traffic monitoring, home monitoring and manufacturing.

The unique aspects of sensor network can best be examined with significant numbers of prototype devices explicitly designed for this purpose, as opposed to generic computing platforms. Some of the unique requirements for wireless sensing network include:

- Small, lightweight form factor

- Robustness to wide temperature ranges and other demanding environmental conditions
- Battery or other stand-alone power sources
- Low power operation and access to internal power control mechanisms
- A small, low power radio having sufficient range
- A real-time execution environment
- The ability to write program code in a high level language for rapid algorithm hosting and testing
- Reasonable cost

Sensor network technology is a key technology for the 21st century. Cheap, smart devices with multiple onboard sensors, networked through wireless links and the Internet and deployed in large numbers, provide unprecedented opportunities for measuring and controlling homes, cities, and the environment. In addition, networked micro-sensors provide the technology for a broad spectrum of systems in the defense arena, generating new capabilities for reconnaissance and surveillance as well as other tactical applications.

Current and potential applications of sensor networks include: military sensing, physical security, air traffic control, traffic surveillance, video surveillance, industrial and manufacturing automation, distributed robotics, environment monitoring, and building and structures monitoring. The sensors in these applications may be small or large, and the networks may be wired or wireless. However, ubiquitous wireless networks of micro-sensors probably offer the most potential in changing the world of sensing.

1.2 Technology Trend

Current sensor networks can exploit technologies not available 20 years ago and perform functions that were not even dreamed of at that time. Sensors, processors, and communication devices are all getting much smaller and cheaper[2].

Commercial companies such as Ember, Crossbow, and Sensoria are now building and deploying small sensor nodes and systems. These companies provide a vision of how our daily lives will be enhanced through a network of small, embedded sensor nodes.

Wireless networks based upon IEEE 802.11 standards can now provide bandwidth approaching those of wired networks. At the same time, the IEEE has noticed the low expense and high capabilities that sensor networks offer. The organization has defined the IEEE 802.15 standard for personal area networks (PANs), with “personal networks” defined to have a radius of 5 to 10 m. Networks of short-range sensors are the ideal technology to be employed in PANs. The IEEE encouragement of the development of technologies and algorithms for such short ranges ensures continued development of low-cost sensor nets. Furthermore, increases in chip capacity and processor production capabilities have reduced the energy per bit requirement for both computing and communication. Sensing, computing, and communications can now be performed on a single chip, further reducing the cost and allowing deployment in ever larger numbers[3].

Looking into the future, we predict that advances in MEMS (Micro Electro Mechanical System) technology will produce sensors that are even more capable and versatile. For example, Dust Inc., Berkeley, CA, a company that sprung from the late 1990s Smart Dust research project at the University of California, Berkeley, is building MEMS sensors that can sense and communicate and yet are tiny enough to fit inside a cubic millimeter. A Smart Dust optical mote uses MEMS to aim sub-millimeter-sized mirrors for communications. Smart Dust sensors can be deployed using a 3x10 mm "wavelet" shaped like a maple tree seed and dropped to float to the ground. A wireless network of these ubiquitous, low-cost, disposable micro-sensors can provide close-in sensing capabilities in many applications[4].

1.3 Purpose of Project

While sensor networks for various applications may be quite different, they share common technical issues. Therefore, one well-designed sensor network system can be a great helper to variety of sensor network application. The purpose of this project is to develop reusable type of sensor network system that allows us to instrument, observe, and respond to phenomena in the natural environment.

Since sensor network is the mixture of sensing, computing and communication infrastructure, it generally poses considerable technical problems in data processing, communication, and sensor management. Followings are common technical challenges in sensor network implementation.

1.3.1 Network Control and Routing

The network must deal with resources - energy, bandwidth, and the processing power - that are dynamically changing, and the system should operate autonomously, changing its configuration as required. Since there is no planned connectivity in ad hoc networks, connectivity must emerge as needed from the algorithms and software. Since communication links are unreliable and shadow fading may eliminate links, the software and system design should generate the required reliability. This requires research into issues such as network size or the number of links and nodes needed to provide adequate redundancy. Also, for networks on the ground, RF transmission degrades with distance much faster than in free space, which means that communication distance and energy must be well managed. Protocols must be internalized in design and not require operator intervention.

Alternative approaches to traditional Internet methods, including mobile IP, are needed. One of the benefits of not requiring IP addresses at each node is that one can deploy network devices in very large numbers. Also, in contrast to the case of IP, routes are built up from geo-information, on an as-needed basis, and optimized for survivability and energy. This is a way to form connections on demand, for data-specific or application-specific purposes. IP is not likely to be a viable candidate in this context, since it needs to maintain routing tables for the global topology, and because updates in a dynamic sensor network environment incur heavy overhead in terms of time, memory, and energy.

Survivability and adaptation to the environment are ensured through deploying an adequate number of nodes to provide redundancy in paths, and algorithms to find the right paths. Diffusion routing methods, which rely only upon information at neighboring nodes, are a way to address this, although such methods may not achieve the information-theoretic capacity of a spatially distributed wireless network. Another important design issue is the investigation of how system parameters such as network size and density of nodes per square mile affect the tradeoffs between latency, reliability, and energy.

1.3.2 Collaborative Signal and Information Processing

The nodes in an ad hoc sensor network collaborate to collect and process data to generate useful information. Collaborative signal and information processing over a network is a new area of research and is related to distributed information fusion. Important technical issues include the degree of information sharing between nodes and how nodes fuse the information from other nodes. Processing data from more sensors generally results in better performance but also requires more communication resources (and, thus, energy). Similarly, less information is lost when communicating information at a lower level (e.g., raw signals), but requires more bandwidth. Therefore, one needs to consider the multiple tradeoffs between performance and resource utilization in collaborative signal and information processing using micro-sensors[1].

When a node receives information from another node, this information has to be combined and fused with local information. Fusion approaches range from simple rules of picking the best result to model-based techniques that consider how the information is generated. Again there is a tradeoff between performance and robustness. Simple fusion rules are robust but suboptimal while more sophisticated and higher performance fusion rules may be sensitive to the underlying models. In a networked environment, information may arrive at a node after traveling over multiple paths. The fusion algorithm should recognize the dependency in the information to be fused and avoid double counting. Keeping track of data pedigree is an approach used in networks with large and powerful sensor nodes, but this approach may not be practical for ad hoc networks with limited processing and communication resources.

Sensor networks are frequently used in the detection, tracking, and classification of targets. Data association is an important problem when multiple targets are present in a small region. Each node must associate its measurements of the environment with individual targets. In addition, targets detected by one node have to be associated with targets detected by other nodes to avoid duplication and enable fusion. Optimal data association is computationally expensive and requires significant bandwidth for communication. Thus distributed data association is also a tradeoff between performance and resource utilization, requiring distributed data association algorithms tailored to sensor nets.

Other processing issues include how to meet mission latency and reliability requirements, and how to maximize sensor network operational life. A dense network of cheap sensors may allow spatial sampling without the need for expensive algorithms. These algorithms must be asynchronous, as the processor speeds and communication capabilities may vary or even disappear and reappear.

Sensor nodes must determine results with progressively increasing accuracy, and so the processes can be terminated when enough precision is gained.

1.3.3 Tasking and Querying

A sensor field is like a database with many unique features. Data is dynamically acquired from the environment, as opposed to being entered by an operator. The data is distributed across nodes, and geographically dispersed nodes are connected by unreliable links. These features render the database view more challenging, particularly for military applications given the low-latency, real-time, and high-reliability requirements of the battlefield.

It is important that users have a simple interface to interactively task and query the sensor network. An example of a human-network interface is a handheld unit that accepts speech input. The users should be able to command access to information, e.g., operational priority and type of target, while hiding details about individual sensors. One challenge is to develop a language for querying and tasking, as well as a database that can be readily queried. Other challenges include finding efficient distributed mechanisms for query and task compilation and placement, data organization, and caching.

Mobile platforms can carry sensors and query devices. As a result, seamless internetworking between mobile and fixed devices in the absence of any infrastructure is a critical and unique requirement for sensor networks. For example, an airborne querying device could initiate a query, and then tell the ground sensor network that it will be flying over a specific location after a minute, where the response to the query should be ex-filtrated.

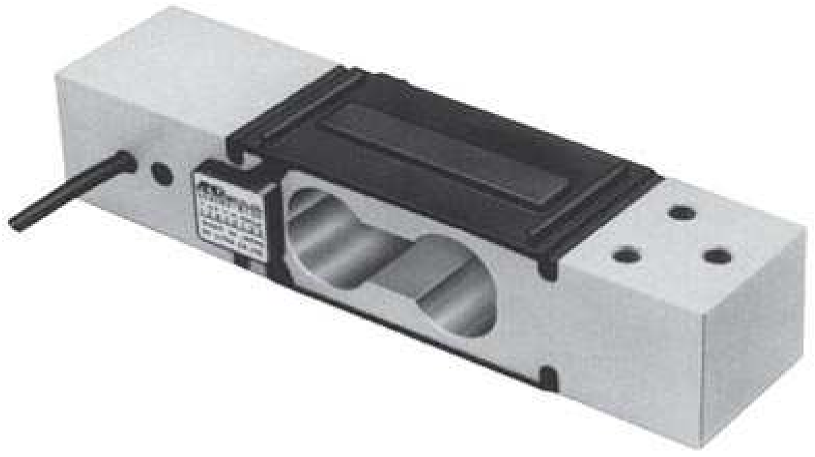
Chapter 2 Hardware Architecture

2.1 Sensor Module

Sensor module has three major parts: strain gauge sensor for each channel, A/D converting part which amplifies and filters analog signal of sensor into digital output, and controlling part which controls each channel for data collection and sends collected data to communication part.

2.1.1 Beam Load Cell

Figure 1. LC4103 Beam Load Cell Strain Gauge[13]



This beam load cell has been mainly used for single point platform weighing.

Its produced output signal is directly proportional to the applied weight. Since the single point design is highly resistant to eccentric loading, the scale base and weighing platform could be directly mounted conveniently. Following is its specification[5].

Table 1. LC4103 Beam Load Cell Specification[13]

Rated Output	1.0mV/V±15/-0%
Error Range	±0.015% of R.O.
Safe Overload	300% of R.O.
Zero balance	20 ± 5% of R.O.
Compensated temperature range	-10℃ - +40℃
Suggested voltage	12V
Maximum voltage	15V
Input terminal resistance	Approx. 400Ω
Output terminal resistance	350Ω ± 5Ω
Platform size	400×600㎜
Line length	2m

2.1.2 Instrumentation Amplifier

The differential input single-ended output instrumentation amplifier can be selected as one of the most versatile signal processing amplifiers available. It precisely amplifies differential dc or ac signals while rejects large values of common mode noise. By adapting integrating circuits, high level of performance is obtained without incurring the cost problem[6].

Figure 2. Amplification Circuit Layout

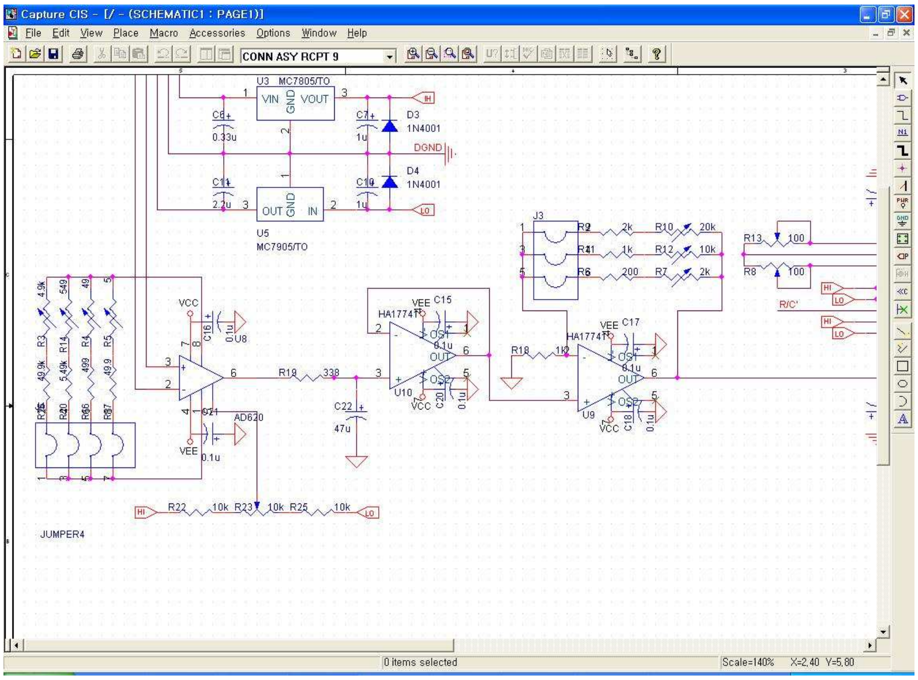


Figure 3. Three Op Amp IA Design[6]

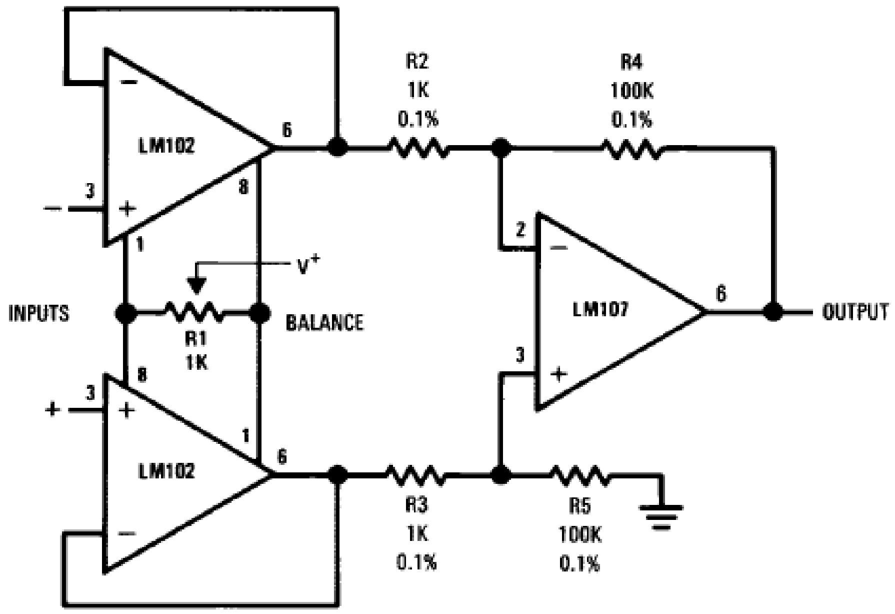
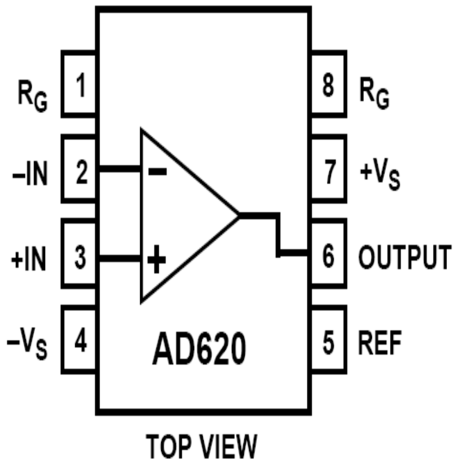


Figure 3 show a basic instrumentation amplifier which provides a 10 volt output for 100 mW input, while rejecting greater than +/- 11V of common mode noise. The input signal should be buffered two voltage followers to obtain good input characteristics. The LM102 has 10,000 M Ω input impedance with 3 nA input current. The high level of input impedance provides two benefits. First it makes the instrumentation amplifier possibly being used with high source resistances while still maintaining low error. Secondly it allows the source resistances to be unbalanced by over 10,000 Ω with no degradation in common mode rejection. The followers providing gain and rejecting the common mode voltage guide the operation of a balanced differential amplifier, as show in Figure 3. By setting the ratio of R4 to R2 and R5 to R3, the gain could be decided. Calculating with the values shown in Figure 3, the gain for differential signals is 100[6].

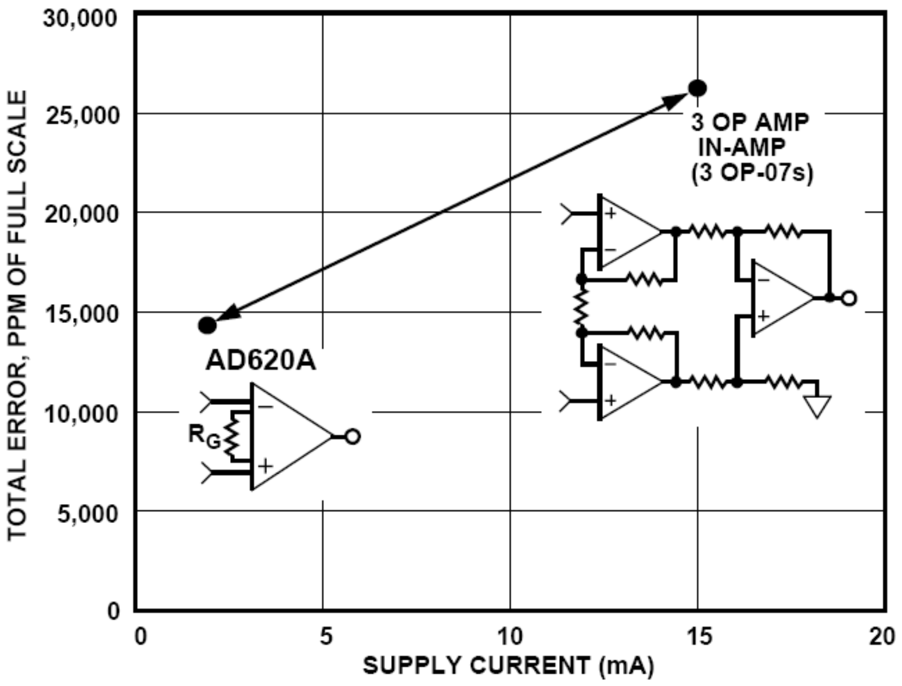
Figure 4. AD620 Connection Diagram[6]



The AD620 can set gains of 1 to 10,000 with only one external resistor. Therefore, it can be called as a low cost, high accuracy instrumentation amplifier. Furthermore, the AD620 features 8-lead SOIC and DIP packaging is smaller than discrete designs. It offers lower power (only 1.3 mA max supply current). With these features, the AD620 can be a good fit for battery-powered, portable (or remote) applications.

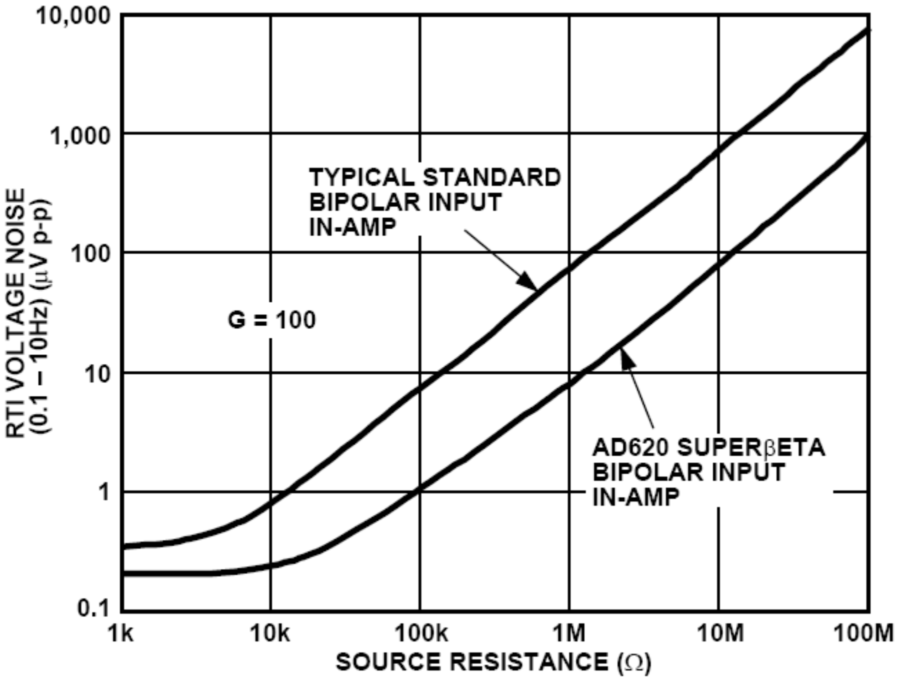
The AD620 has high accuracy of 40 ppm maximum nonlinearity, low offset voltage of 50 μV max, and offset drift of 0.6 μV/° C max. To consider these features, the AD620 is quite useful in precision data acquisition systems, such as weigh scales and transducer interfaces. To consider more on the features of AD620 like the low noise, low input bias current, and low power, it is not difficult to find that AD620 is used in medical applications, such as ECG and noninvasive blood pressure monitors[6].

Figure 5. Three OP Amp IA Designs vs. AD620[6]



With the use of Superbeta processing in the input stage, the low input bias current of 1.0 nA max can be made. Due to its low input voltage noise of 9 nV/√Hz at 1 kHz, 0.28 μV p-p in the 0.1 Hz to 10 Hz band, and 0.1 pA/√Hz input current noise, the AD620 also works well as a preamplifier. To consider its settling time of 15 μs to 0.01%, the AD620 is enough to fit into multiplexed applications, and its low cost fulfills the requirements for the designs with one in-amp per channel[6].

Figure 6. AD620 Total Voltage Noise vs. Source Resistance[6]



Instrumentation amplifiers like the AD620 offer high CMR. CMR is a measure of the change in output voltage when both inputs are changed by equal amounts. Implementing a full-range input voltage change and a specified source imbalance, these specifications are usually required.

To maintain optimal CMR level, the reference terminal should be tied to a low impedance point. Differences in capacitance and resistance also should be kept to a minimum between the two inputs. In many applications, shielded cables are adopted for noise minimization. For best CMR over frequency, the shield should be properly driven. Figure 7 and Figure 8 show active data guards that are configured to improve ac common-mode rejections. The capacitance mismatch between the inputs can be minimized by bootstrapping the capacitances of input cable shields.

Figure 7. Differential Shield Driver[6]

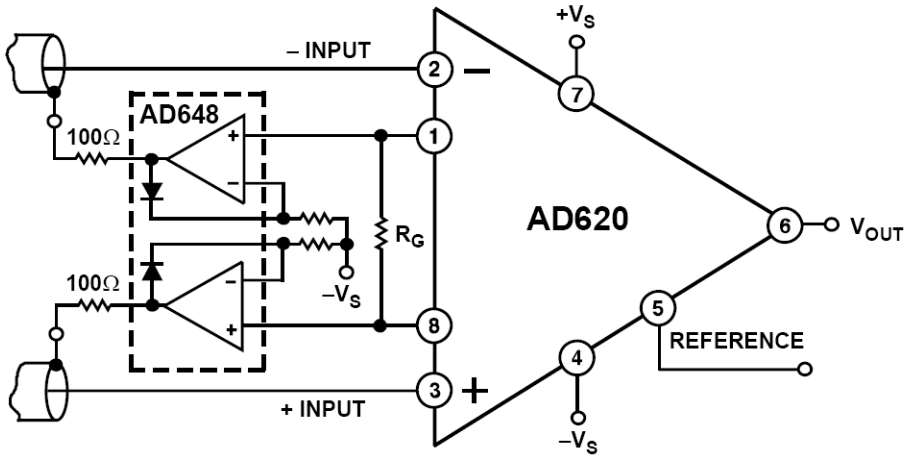
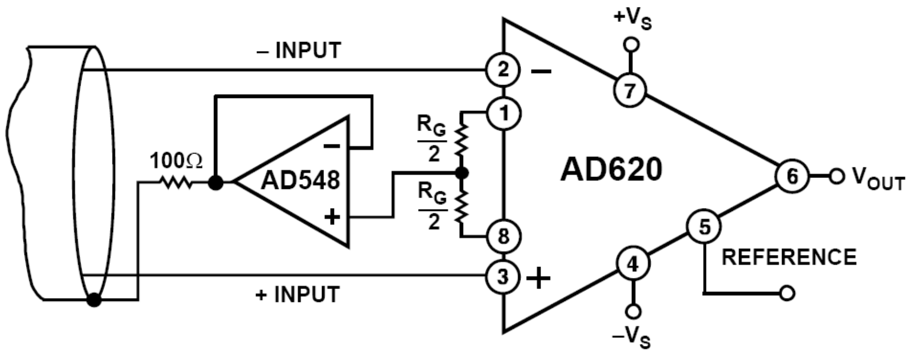


Figure 8. Common-Mode Shield Driver[6]



2.1.3 Operational Amplifier

The term operational amplifier or op-amp refers to a class of high-gain DC coupled amplifiers with two inputs and a single output. The modern integrated circuit version is typified by the famous 741 op-amp. Some of the general characteristics of the IC version are high gain, on the order of a million, high input impedance, low output impedance, used with split supply, usually +/- 15V, used with feedback, with gain determined by the feedback network.

The most common and most famous op-amp is the mA741C or just 741, which is packaged in an 8-pin mini-DIP.

Figure 9. HA17741 Pin Arrangement[6]

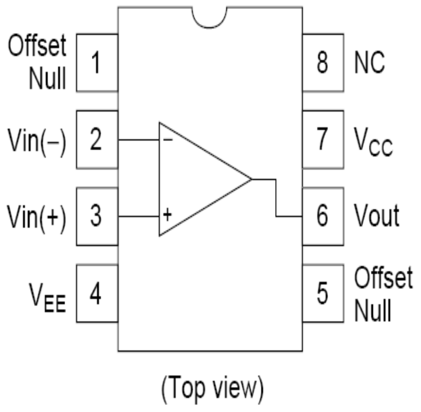
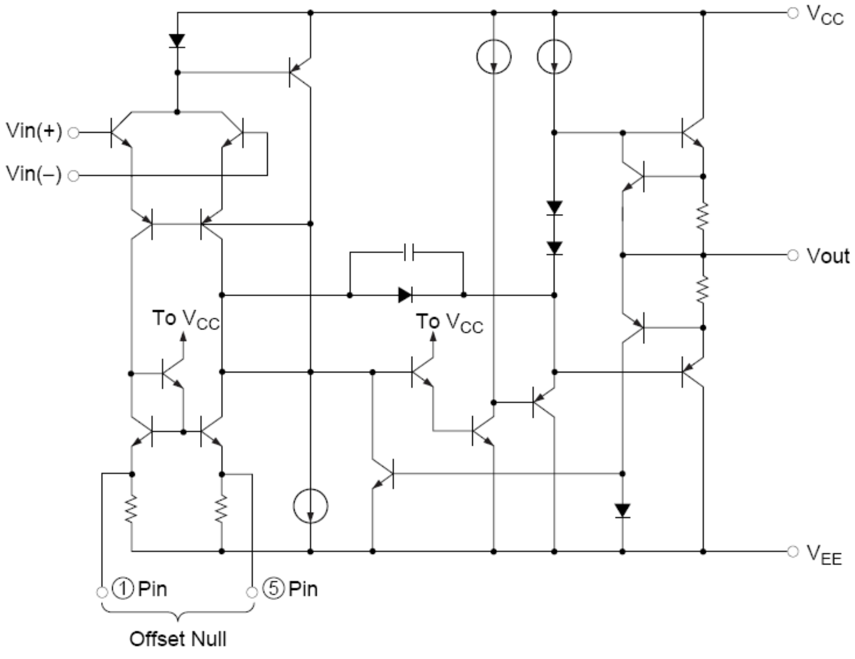


Figure 10. HA17741 Circuit Structure[6]

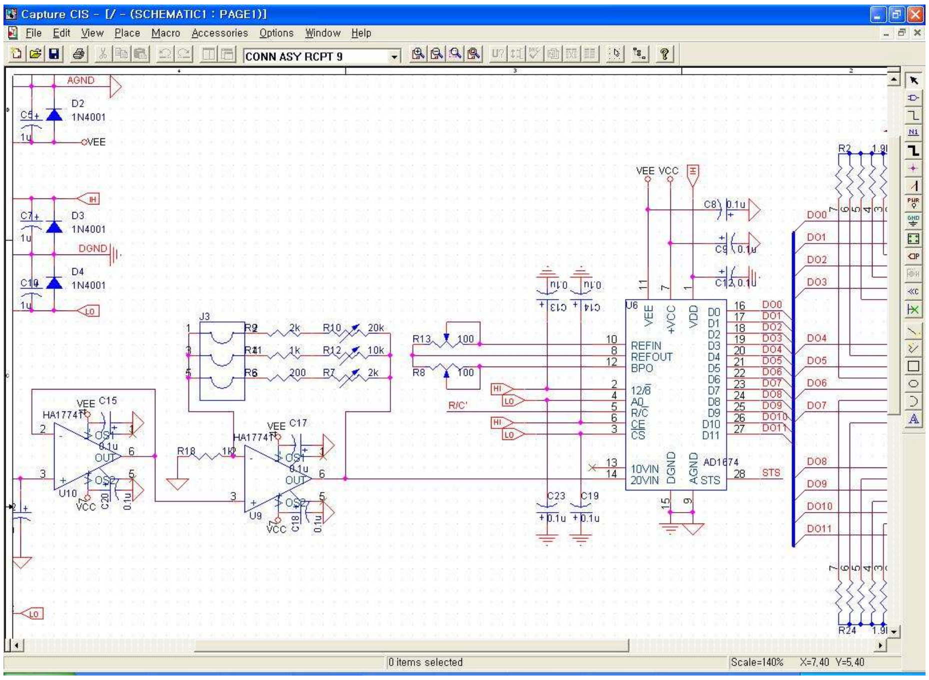


2.1.4 A/D Converter

The AD1674 is a complete, multipurpose, 12-bit analog-to-digital converter. It has a user-transparent onboard sample-and-hold amplifier (SHA), 10 volt reference, clock and three-state output buffers for microprocessor interface[7].

The AD1674 is pin compatible with the industry standard AD574A and AD674A. While delivering faster conversion rate, it also has a sampling function. The on-chip SHA delivers a wide input bandwidth supporting 12-bit accuracy over the full Nyquist bandwidth of the converter.

Figure 11. A/D Converter Circuit Layout

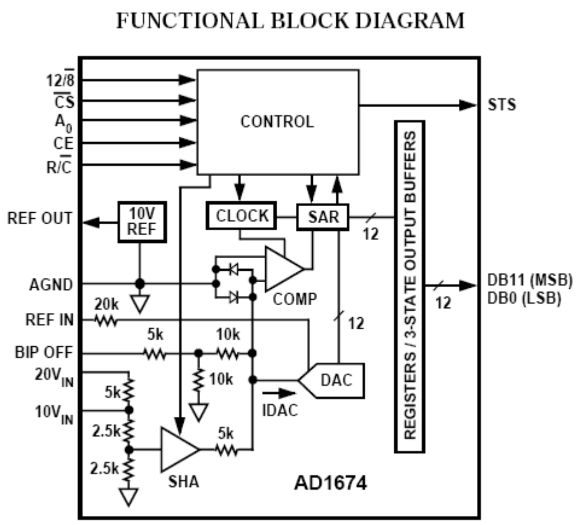

Capture CIS - [/ - (SCHEMATIC1 : PAGE1)]
File Edit View Place Macro Accessories Options Window Help
CONN ASY RCPT 9
AD1674
HA1774T
D2 1N4001
D3 1N4001
D4 1N4001
0 items selected
Scale=140% X=7.40 Y=5.40

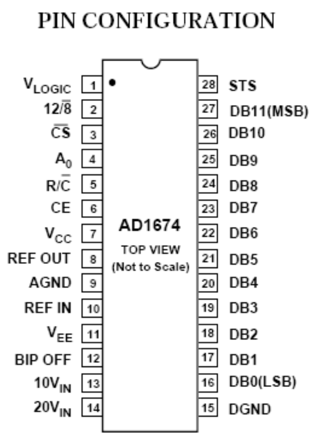
Not only ac parameters (such as S/(N+D) ratio, THD, and IMD) but also dc parameters (offset, full-scale error, etc.) is fully specified by the AD1674. The AD1674 is ideal for use in signal processing and traditional dc measurement applications since its support for both ac and dc specifications.

In order to combine high performance bipolar analog circuitry into the same die with digital CMOS logic, The AD1674 design adapted Analog Devices' BiMOS II process.

According to temperature grades, there are five available grades. For operation over the 0° C to +70° C temperature range, the AD1674J and K grades are suitable. The A and B grades are for the temperature from -40° C to +85° C. Finally the AD1674T grade is specified from -55° C to +125° C. The J and K grades are available in both 28-lead plastic DIP and SOIC. The A and B grade devices are available in 28-lead hermetically sealed ceramic DIP and 28-lead SOIC. The T grade is available in 28-lead hermetically sealed ceramic DIP[7].

Figure 12. AD1674 Functional Block Diagram, Pin Configuration[7]



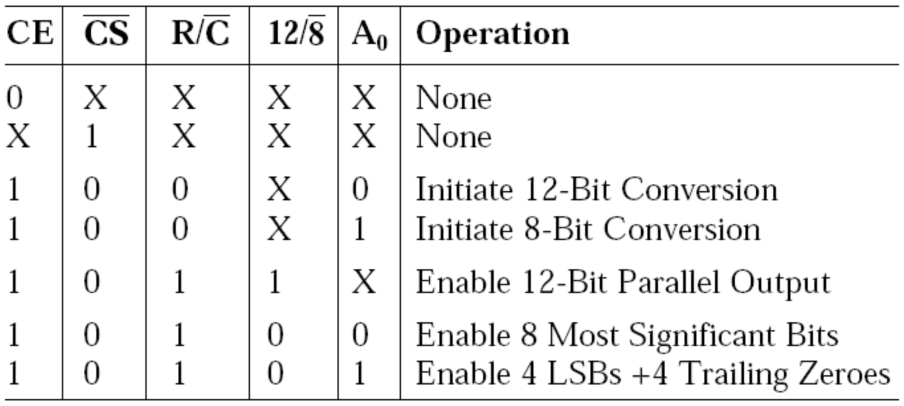


The AD1674 is a complete 12-bit, 10 ms sampling analog-to-digital converter. A block diagram of the AD1674 is shown on Figure 12.

In order to initiate a conversion, AD1674 commands the control section placing the sample-and-hold amplifier (SHA) in the hold mode, enabling the clock, and resetting the successive approximation register (SAR). During the conversion cycle, its process cannot be stopped or restarted. Therefore, data is not available from the output buffers. When the conversion has been completed, the SAR, timed by the internal clock, will sequence through the conversion cycle and return an end-of-convert flag to the control section. The control section will then disable the clock. The SHA will be switched to sample mode. In order to acquire 12-bit degree accuracy, the STS LOW going edge should be delayed. During the SHA acquisition interval, external command can ask the control section data read functions anytime.

During the conversion cycle, the SAR sequences the internal 12-bit, 1 mA full-scale current output DAC from the most significant bit (MSB) to the least significant bit (LSB). Its output accurately balances the current through the 5 kW resistor from the input signal voltage held by the SHA. While maintaining a 1 mA full-scale output current through the 5 kW resistor for both ranges, the SHA’s input scaling resistors divide the input voltage by 2 for the 10 V input span and by 4 V for the 20 V input span, Following is the comparing process determining whether the addition of each successively weighted bit current causes the DAC current sum to be greater than or less than the input current. If the sum is less, the bit is left on; if more, the bit is turned off. After completing all bits test, the SAR contains a 12-bit binary code. The code accurately represents the input signal to within ±1/2 LSB[7].

Table 2. AD1674 Truth Table[7]

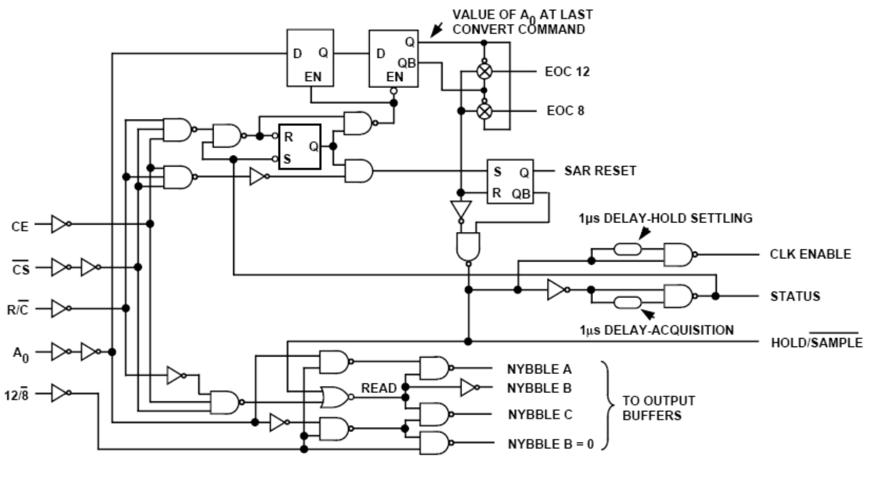


CE	$\overline{CS}$	$R/\overline{C}$	$12/\overline{8}$	A_0	Operation
0	X	X	X	X	None
X	1	X	X	X	None
1	0	0	X	0	Initiate 12-Bit Conversion
1	0	0	X	1	Initiate 8-Bit Conversion
1	0	1	1	X	Enable 12-Bit Parallel Output
1	0	1	0	0	Enable 8 Most Significant Bits
1	0	1	0	1	Enable 4 LSBs +4 Trailing Zeroes

There are two operating mode in The AD1674, the full control mode and the stand-alone mode. First, the full-control mode utilizes all the AD1674 control signals. In this mode, through a single data bus, multiple devices can be addressed simultaneously. On the other hand, the stand-alone mode is specialized in systems with dedicated input ports available and thus not requiring full bus interface capability.

Table 2 is a truth table for the AD1674, and Figure 13 illustrates the internal logic circuitry.

Figure 13. AD1674 Equivalent Internal Logic Circuitry[7]



Following are related circuit diagrams.

Figure 14. Regulator Circuit Layout

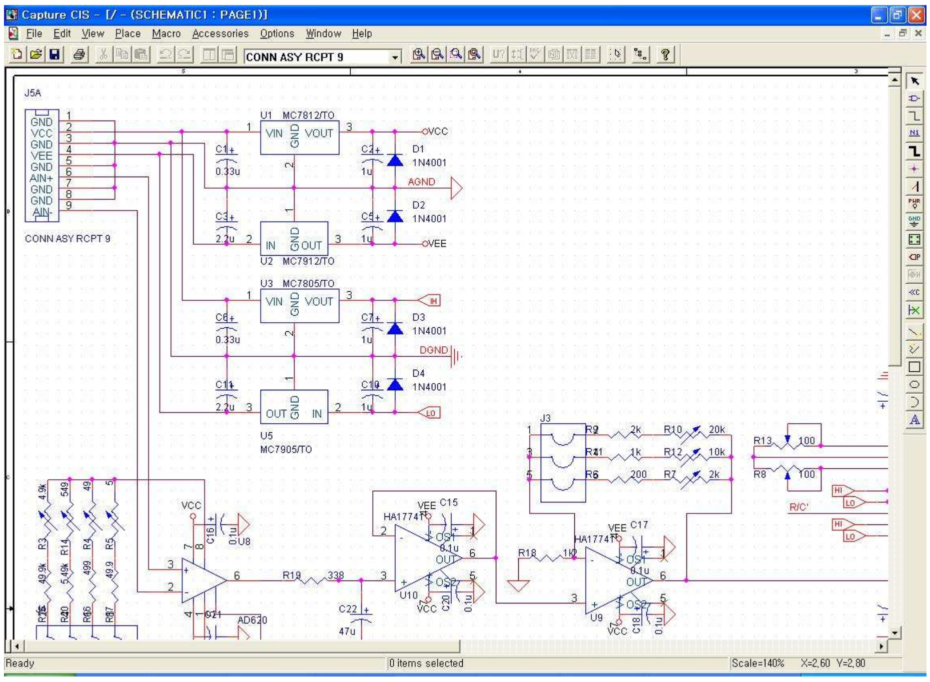


Figure 15. Total Circuit Layout

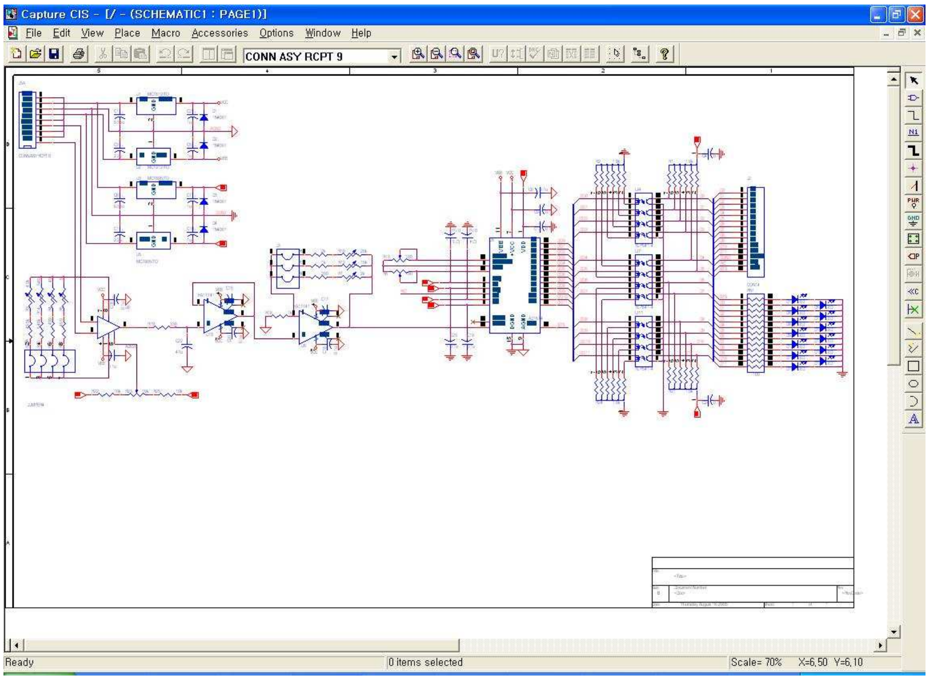

Capture CIS - [/ - (SCHEMATIC1 : PAGE1)]
File Edit View Place Macro Accessories Options Window Help
CONN ASY RCPT 9
Ready
0 items selected
Scale= 70%
X=6,50 Y=6,10

Figure 16. A/D Converter Top Layout

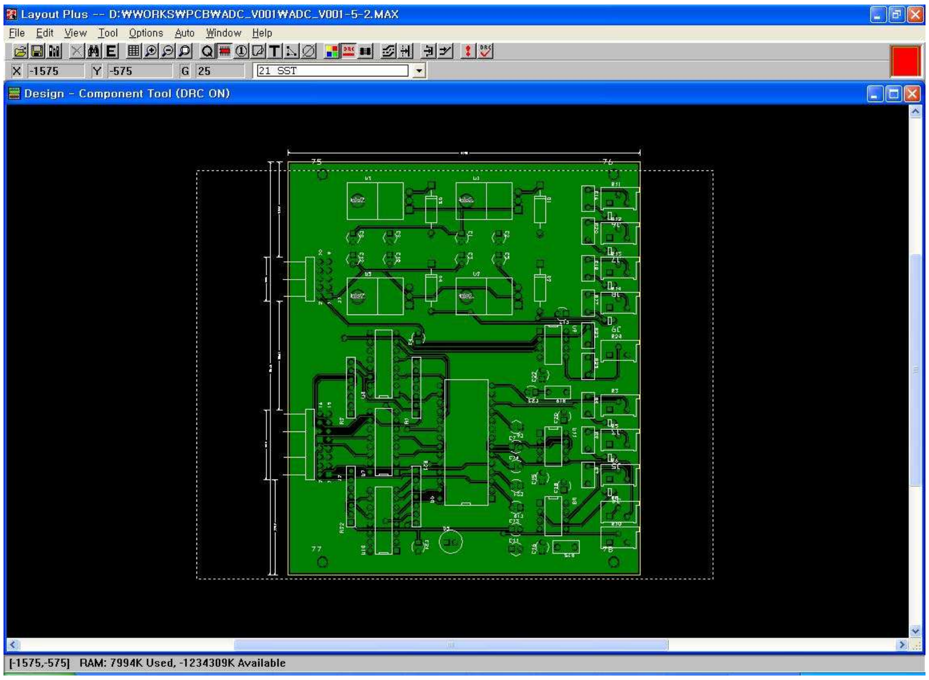

Layout Plus -- D:₩WORKS₩PCB₩ADC_V001₩ADC_V001-5-2.MAX
File Edit View Tool Options Auto Window Help
X -1575 Y -575 G 25 21 SST
Design - Component Tool (DRC ON)
[-1575,-575] RAM: 7994K Used, -1234309K Available

Figure 17. A/D Converter Bottom Layout

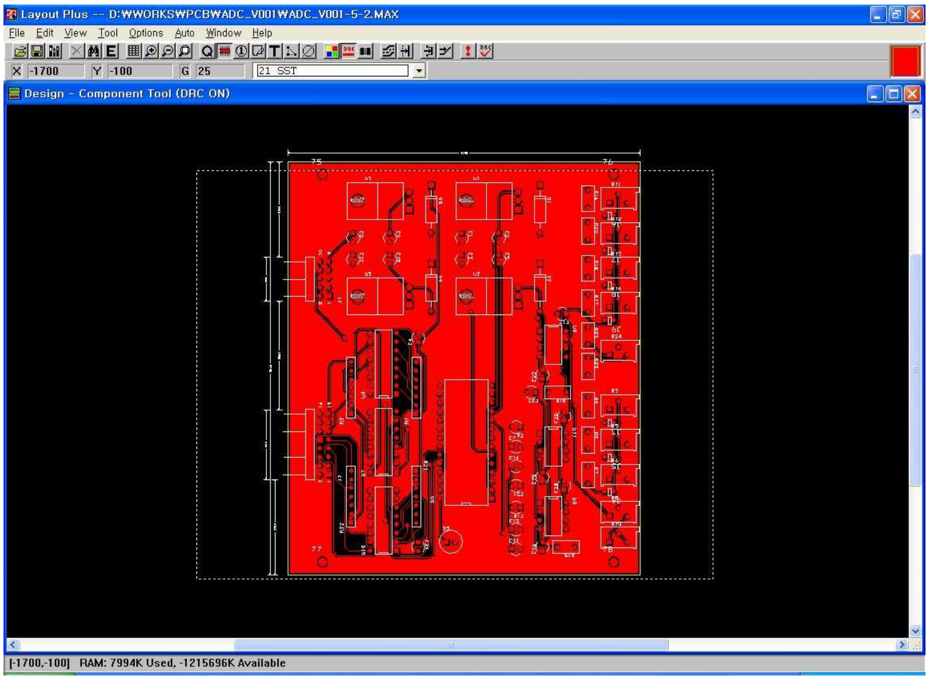

Layout Plus -- D:\WORKS\PCB\ADC_V001\ADC_V001-5-2.MAX
File Edit View Tool Options Auto Window Help
X -1700 Y -100 G 25 21 SST
Design - Component Tool (DRC ON)
[-1700,-100] RAM: 7994K Used, -1215696K Available

Figure 18. Controller Top Layout

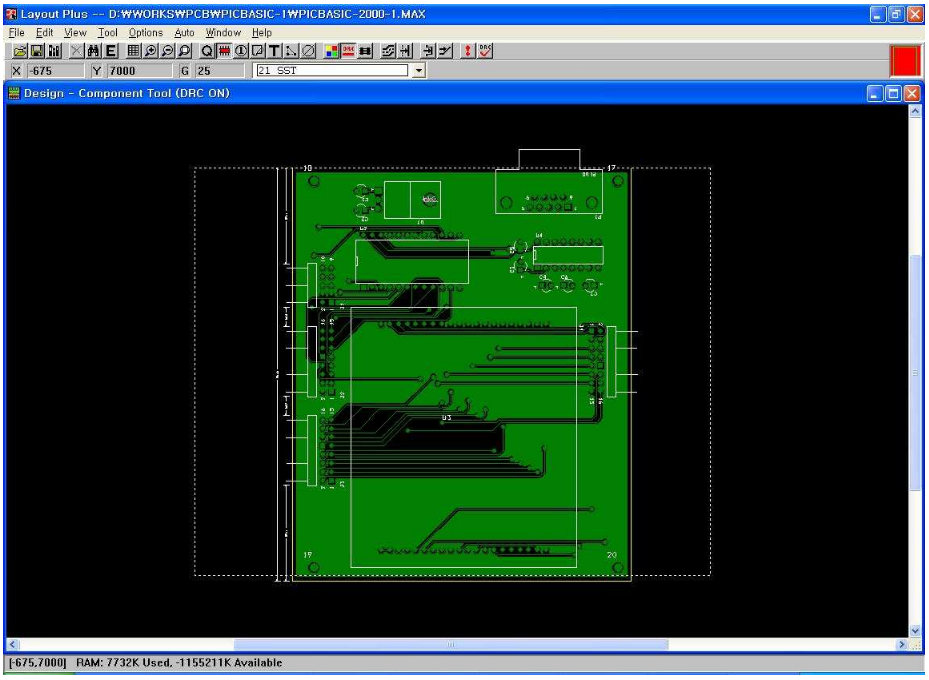

Layout Plus -- D:￦WORKS￦PCB￦PICBASIC-1￦PICBASIC-2000-1.MAX
File Edit View Tool Options Auto Window Help
X -675 Y 7000 G 25
21 SST
Design - Component Tool (DRC ON)
[-675,7000] RAM: 7732K Used, -1155211K Available

Figure 19. Controller Bottom Layout

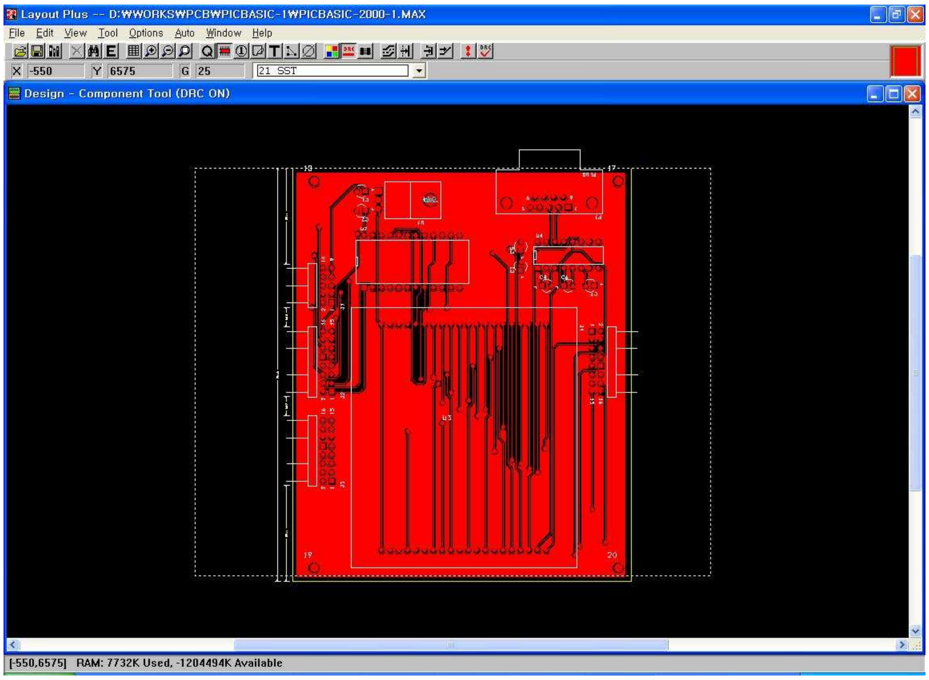
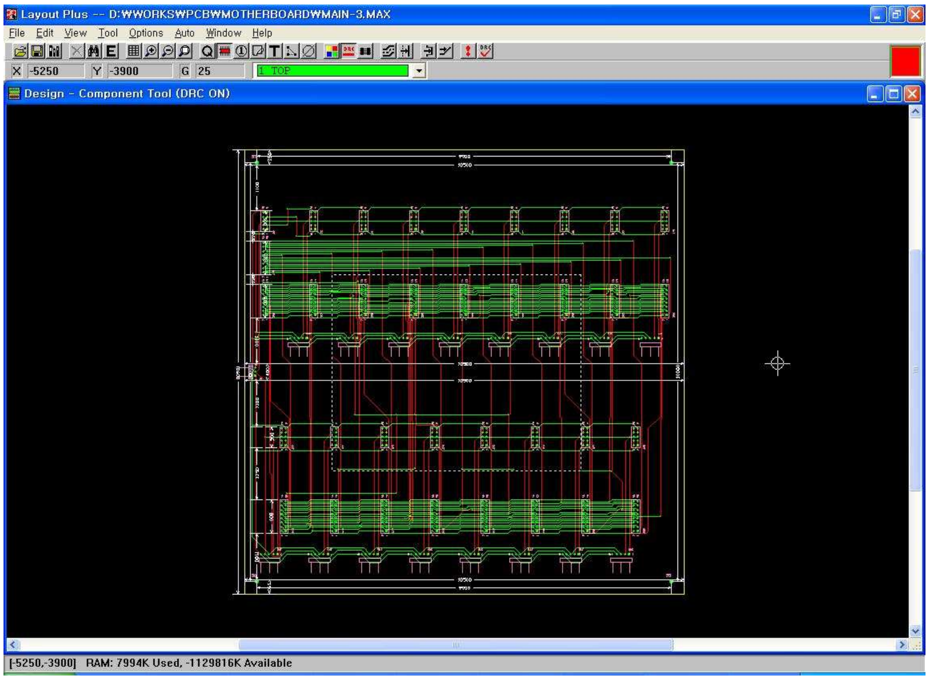

Layout Plus -- D:₩WORKS₩PCB₩PICBASIC-1₩PICBASIC-2000-1.MAX
File Edit View Tool Options Auto Window Help
X -550 Y 6575 G 25
21 SST
Design - Component Tool (DRC ON)
[-550,6575] RAM: 7732K Used, -1204494K Available

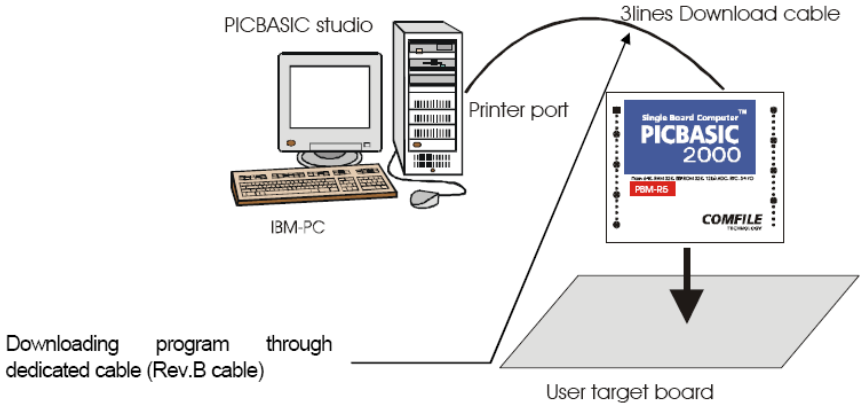
Figure 20. Mainboard Layout


Layout Plus -- D:₩WORKS₩PCB₩MOTHERBOARD₩MAIN-3.MAX
File Edit View Tool Options Auto Window Help
X -5250 Y -3900 G 25
1 TOP
Design - Component Tool (DRC ON)
[-5250,-3900] RAM: 7994K Used, -1129816K Available

2.2 Processor Module

Simply PICBASIC 2000 is micro single board computer which can be applicable in various field of automation control area such as automated machine, thermal controller, inspection ZIG, ROBOT controller, data acquaintance device etc. Based on BASIC language, the PICBASIC language has various useful single instructions such as ADIN, PWN, and SEROUT etc. Since it take the printer port interface of PC as programming channel, programming work could be more simple and easy[10].

Figure 21.PICBASIC Programming Environment[10]

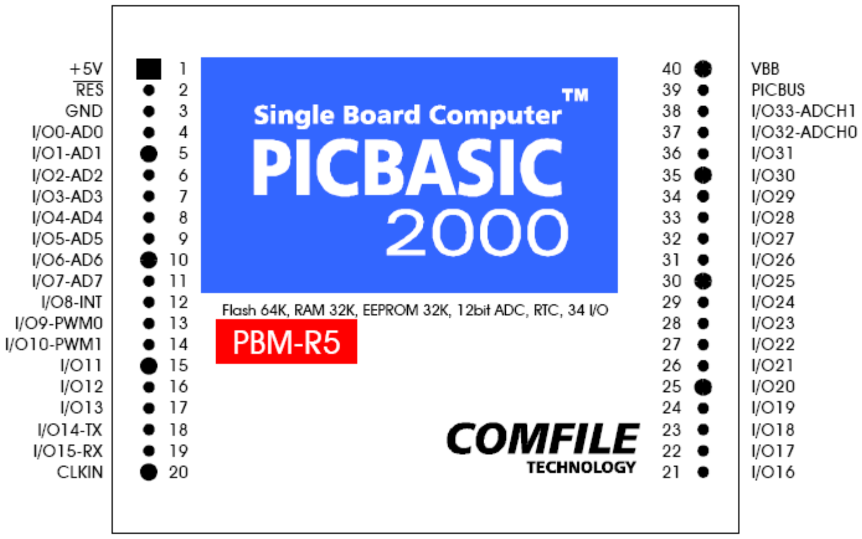


FLASH ROM 29C512 (64Kbyte) is used for the main memory of PBM-R5 module. SRAM 62256 (32Kbyte) and a EEPROM 24LC256 (32Kbyte) are both for its data memory[10].

Table 3. PBM-R5 Module Specification[10]

Main memory	64K byte
Data memory	32K byte
I/O ports	34
Main processor	PIC16F877
Oscillation frequency	20MHz
Memory access	Serial
EEPROM for data	32K byte
No. of Pin	40
A/D channel (Resolution)	8 (10 bit)
12 bit A/D	2 channel
PWM channel (Resolution)	2 (10 bit)
String	Available
32 bit integer, real number	Available
RE232 buffering method	Available

Figure 22. PBM-R5 Pinout[10]



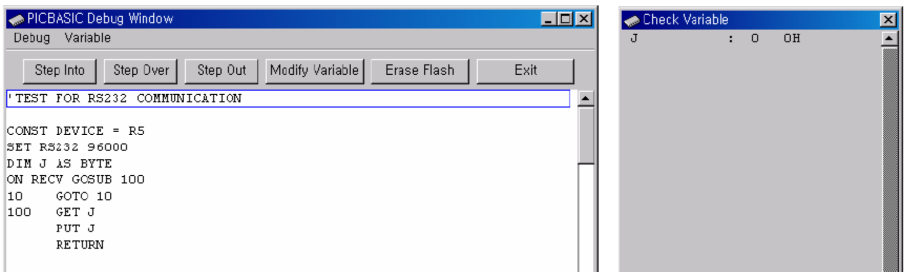
Pin No.	Description		Port block	Input	Function
1	+5V	Power,5V			
2	/RES	Reset,5V			
3	GND	Ground			
4	I/00/AD0	Port 0	Block0	TTL	10bits AD input
5	I/01/AD1	Port 1	Block0	TTL	10bits AD input
6	I/02/AD2	Port 2	Block0	TTL	10bits AD input
7	I/03/AD3	Port 3	Block0	TTL	10bits AD input
8	I/04/AD4	Port 4	Block0	TTL	10bits AD input
9	I/05/AD5	Port 5	Block0	TTL	10bits AD input
10	I/06/AD6	Port 6	Block0	TTL	10bits AD input

11	I/07/AD7	Port 7	Block0	TTL	10bits AD input
12	I/08/INT	Port 8	Block1	ST	Edge interrupt
13	I/09/PWM0	Port 9	Block1	ST	10bitsPWMport
14	I/010/PWM1	Port 10	Block1	ST	10bitsPWMport
15	I/011	Port 11	Block1	ST	
16	I/012	Port 12	Block1	ST	
17	I/013	Port 13	Block1	ST	
18	I/014/TX	Port 14	Block1	ST	RS232C trans
19	I/015/RX	Port 15	Block1	ST	RS232C recv
20	CLKIN	Count IN		ST	
21	I/016	Port 16	Block2	ST	
22	I/017	Port 17	Block2	ST	
23	I/018	Port 18	Block2	ST	
24	I/019	Port 19	Block2	ST	
25	I/020	Port 20	Block2	ST	
26	I/021	Port 21	Block2	ST	
27	I/022	Port 22	Block2	ST	
28	I/023	Port 23	Block2	ST	
29	I/024	Port 24	Block3	ST	
30	I/025	Port 25	Block3	ST	
31	I/026	Port 26	Block3	ST	
32	I/027	Port 27	Block3	ST	
33	I/028	Port 28	Block3	ST	
34	I/029	Port 29	Block3	ST	
35	I/030	Port 30	Block3	ST	

36	I/031	Port31	Block3	ST	
37	I/032/ADC0	Port32	AD only		12bits,ADinput
38	I/033/ACD1	Port 33	AD only		12bits,ADinput
39	PICBUS	LCD port			
40	VBB	RTC	Battery	TM	Terminal

PICBASIC has PICBASIC studio as Integrated Development Environment. By including Compiler, Editor, Debugger and etc into one programming environment, developing process could be simple like a word processing job. It is compatible with Microsoft Windows series and not with MAC and UNIX platform.

Figure 23. PICBASIC Studio Programming Interface[10]



```
'TEST FOR RS232 COMMUNICATION

CONST DEVICE = R5
SET RS232 96000
DIM J AS BYTE
ON RECV GOSUB 100
10     GOTO 10
100    GET J
       PUT J
       RETURN
```

```
J            :  0    0H
```

When programming in PICBASIC, most common interface is very RS232C communication since it is widely used in PC as well as other devices such as card reader, small-size printer, display, etc. PICBASIC language includes various instructions for RS232C such as SERIN, SEROUT, PUT and GET. Furthermore it is worth mentioning its instructions for dealing with receiving interrupt such as ON RECV etc.

In order to get the data from chosen channel, 0V digital signal should be on into A/D converter' s chip selection pin of chosen channel. Each channel of A/C converter could send 12 bits digital signal to AD1674 and Three-state buffer with only one digital signal line. Therefore, two digital signal lines from microprocessor are needed, one for channel selection and one for A/C Conversion. Following is a part of PIC BASIC source code we programmed and some explanations.

List 1. PICBASIC Source Code and Explanations

```
CONST DEVICE = R5           'Device = PBM-R5 Module
SET RS232 38400             'Declare RS232C communication baud rate
                            'as 38400 baud rate

DIM KH AS BYTE              'Declaration of High 8 bits Integer
DIM KH_BUF AS BYTE          'Declaration of High 8 bits Integer Buffer
DIM KL AS BYTE              'Declaration of Low 8 bits Integer
DIM KK AS BYTE              'Declaration of Operation Code

KH = 0                      'Initialization of variables
KL = 0
KH_BUF = 0
KK = 0

ON RECV GOSUB 100           'RS232C Receiving Interrupt
```

```
                         ‘This instruction calls label 100 when RX pin
                         ‘(I/O 15th) receives data while RS232C hardware
                         ‘is used. If RS232C receives data while a
                         ‘program operates, the program stores data in
                         ‘extra buffer after jumping to specific routine.

10 OUT 29, 0
GOTO 10

100 GET KK               ‘Yes, Copy RECV to KK

IF KK = 97 THEN          ‘Operation code = Read Low

KH = 0                   ‘Initialization for routine case
KL = 0
KH_BUF = 0

OUT 29, 0
OUT 12, 1
OUT 13, 1
OUT 31, 1
OUT 30, 0
DELAY 1                  ‘Delay by millisecond for synchronization
                         ‘The range of delay is 0-65535
OUT 29, 1
DELAY 1
```

```
KL = BYTEIN(2)          ‘Input by byte from port block 2
KH = BYTEIN(3)          ‘Input by byte from port block 3
                        ‘This instruction reads eight ports
                        ‘simultaneously and save them at a variable by
                        ‘8bits. Port block is a unit of combined eight
                        ‘ports. R5 module has four port blocks.
OUT 29, 0
DELAY 1

KH_BUF = KH AND &B00001111
                        ‘Take only 4 bits of high bits
                        ‘We do 12bits Conversion
                        ‘L8 + H4
PUT KL                  ‘Output Low 8 bits to RS232C
                        ‘This instruction outputs data through RS232C
                        ‘hardware. The data is outputted through I/O
                        ‘port 12 in the formant of 8bit, none parity and
                        ‘1 stop bit. Before using this instruction, you
                        ‘must execute SET RS232 instruction.
END IF
.
.
.
IF KK = 90 THEN         ‘Operation code = Read high
PUT KH_BUF              ‘Output High 4 bits to RS232C
```

```
END IF
RETURN
```

2.3 Radio Module

The radio module uses Promi-SD202 Bluetooth as a terminal device for wireless serial communication using the Bluetooth technology that is international standard of short range wireless communications. Its communication range can be reached up to 100m. In term of noise, it delivers better quality of communication than standard RS232 cables. The FHSS (Frequency Hopping Spread Spectrum) technique of Bluetooth let SD202 have less radio interference and less danger of hacking in air[9].

Table 4. Bluetooth Serial Adapter SD202 Specification[9]

Description	External type wireless serial adapter
Power Class	Class1
RF Range	Up to 100m
Power Connector	DC plug or 9pin
Power Supply	5V-12V
Serial Interface	RS-232
Applicable Antenna	Stub, Dipole, Patch
Bluetooth Qualified	Fully
Type Approved	TELEC, MIC, CE, FCC
Dimensions(H*W*D)	62.5*31.2*16.3

Figure 24. SD202 Components and Assembly[9]

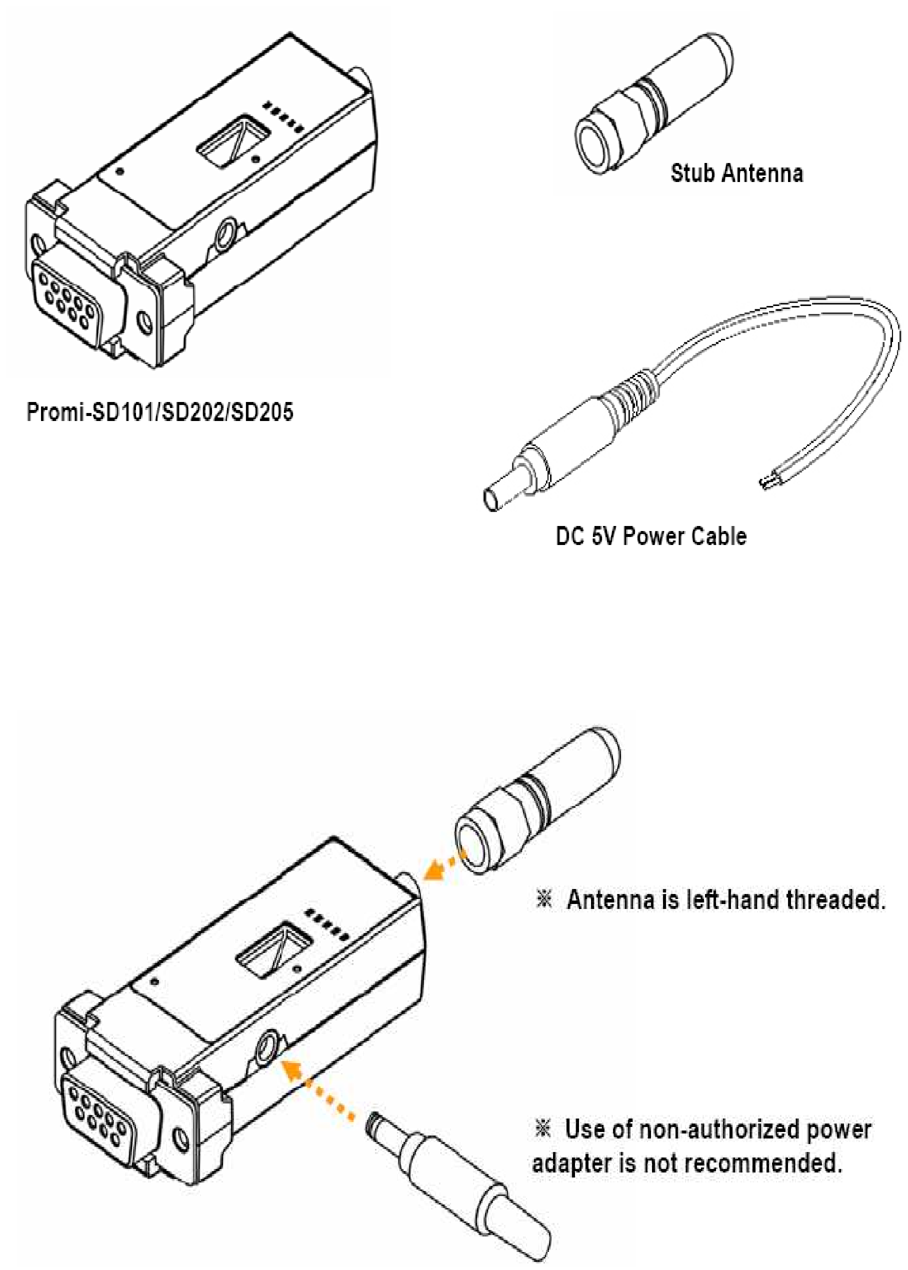
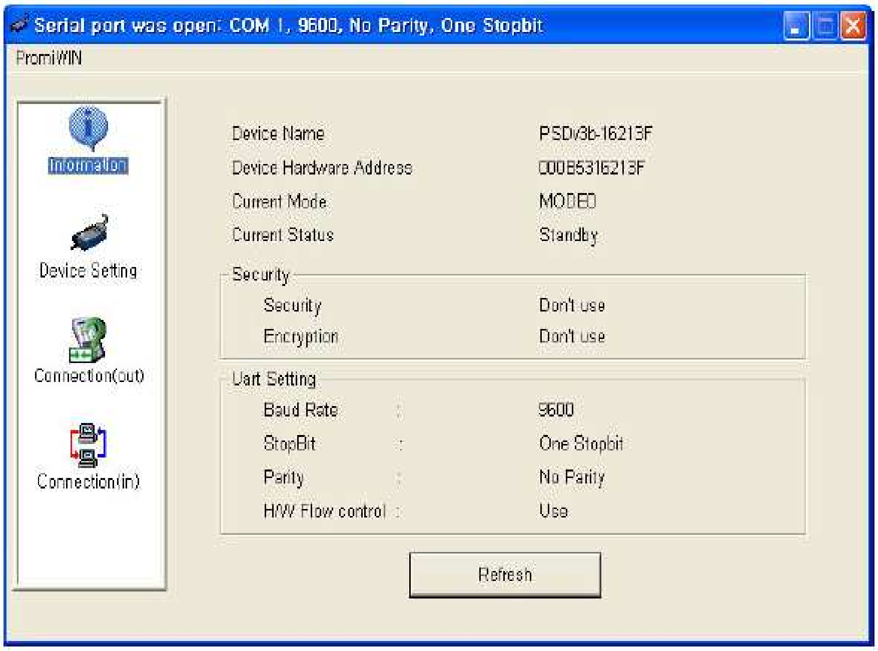

Stub Antenna
Promi-SD101/SD202/SD205
DC 5V Power Cable
※ Antenna is left-hand threaded.
※ Use of non-authorized power adapter is not recommended.

Figure 25. Device Setting Menu[9]



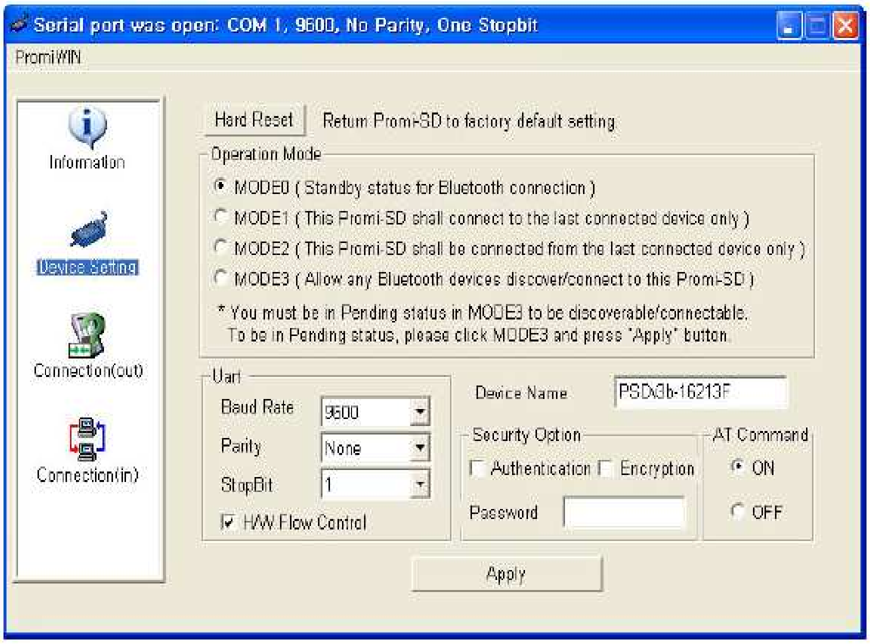
Device Name	PSDv3b-16213F
Device Hardware Address	000B5316213F
Current Mode	MODE0
Current Status	Standby

Security

Security	Don't use
Encryption	Don't use

Uart Setting

Baud Rate	:	9600
StopBit	:	One Stopbit
Parity	:	No Parity
H/W Flow control	:	Use



Hard Reset Return Promi-SD to factory default setting

Operation Mode

- [✓] MODE0 (Standby status for Bluetooth connection)
- MODE1 (This Promi-SD shall connect to the last connected device only)
- MODE2 (This Promi-SD shall be connected from the last connected device only)
- MODE3 (Allow any Bluetooth devices discover/connect to this Promi-SD)

* You must be in Pending status in MODE3 to be discoverable/connectable.
To be in Pending status, please click MODE3 and press 'Apply' button.

Uart

Baud Rate	9600
Parity	None
StopBit	1

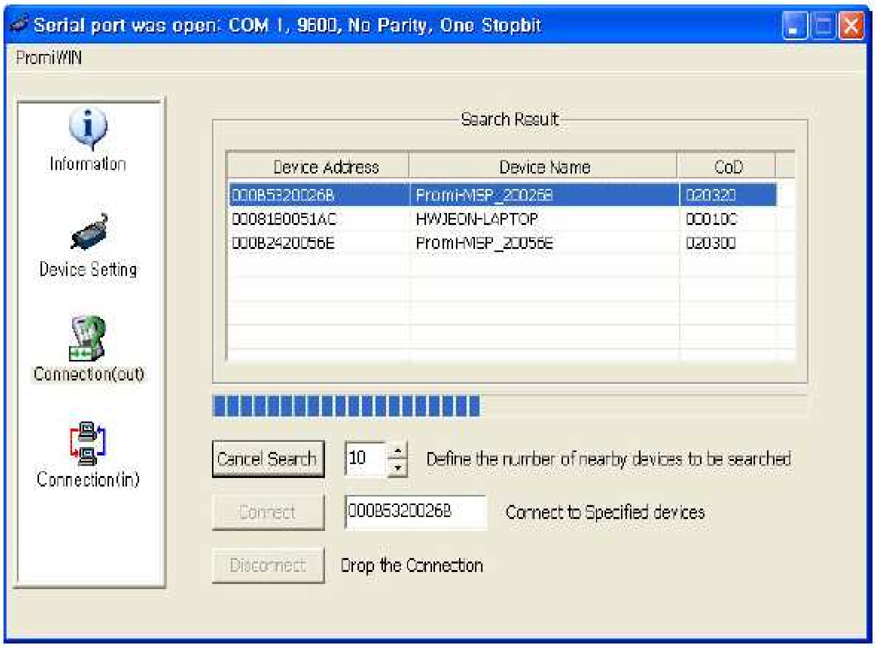
[✓] H/W Flow Control

Device Name PSDv3b-16213F

Security Option: Authentication Encryption

Password

AT Command: [✓ ON] OFF

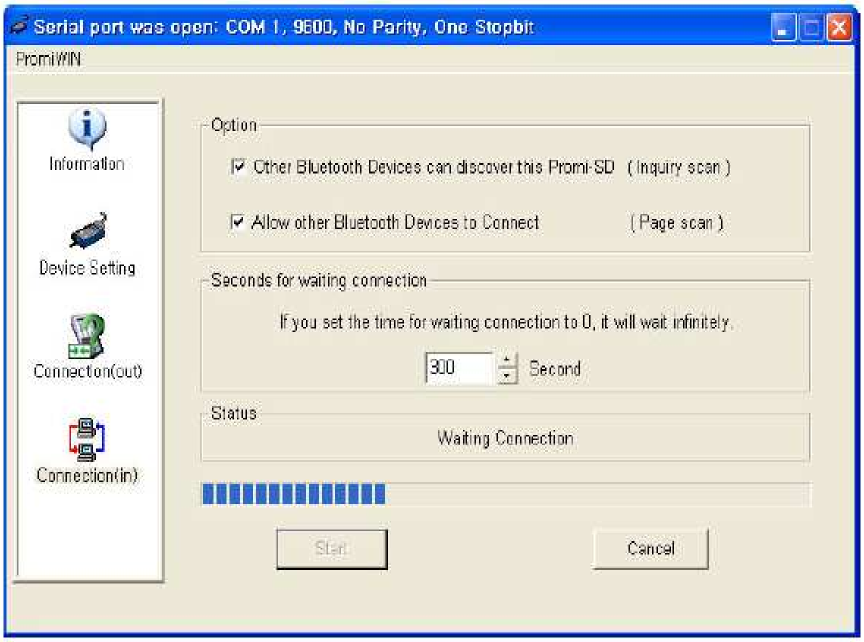


Device Address	Device Name	CoD
000B5320026B	Promi-MSP_20026B	020320
0008160051AC	HWJEON-LAPTOP	00010C
000B2420056E	Promi-MSP_20056E	020300

10 Define the number of nearby devices to be searched

000B5320026B Connect to Specified devices

Drop the Connection



Option

☑ Other Bluetooth Devices can discover this Promi-SD (Inquiry scan)

☑ Allow other Bluetooth Devices to Connect (Page scan)

Seconds for waiting connection

If you set the time for waiting connection to 0, it will wait infinitely.

300 Second

Status

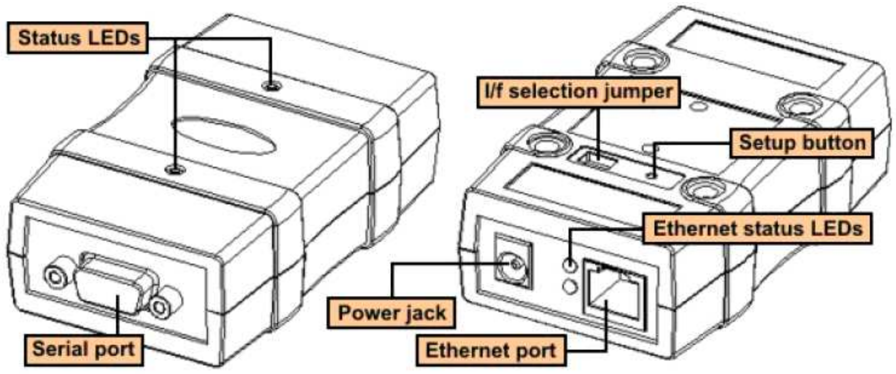
Waiting Connection

2.4 TCP/IP Network Module

Tibbo DS100 device is a Ehternet-to-Serial converting module for external use. Combining one 10BaseT Ethernet port, one serial port and an internal processor, DS100 glues network and serial sides together. The internal processor of DS100 is based on the EM100 Ehternet Module[8].

As an universal hardware platform , the DS100 can run a variety of network and serial communications-related application.

Figure 26. DS100 Connectors and Controls[8]



Power jack of the DS100 fits into large power connectors with 5.5mm diameter. At least 500mA of current rating and 12VDC of nominal voltage should be maintained. The ground of the power jack is located on the outside.

10BaseT-type Ethernet port of the DS100 is working well with all 10BaseT

Ethernet hubs and also 99% of 100BaseT hubs, since most 100BaseT hubs are actually 100/10 devices that auto-detect the type of device connected to each port.

Table 5. DS100 Specification[8]

Ethernet interface	10 BaseT Ethernet
Serial interface and I/O lines	RS232 (TX, RX, RTS, CTS)
Routing Buffers size	510 bytes x 2
Power requirements	DC12V, app. 100mA
Operating temperature	-5 to +70 degrees C
Operating relative humidity	10-90 %
Mechanical dimensions	95x57x30mm
Carton dimensions (bare DS100)	130x100x65mm
Gross weight (bare DS100)	170g
Carton dimensions (DS100-KIT)	320x45x90mm
Gross weight (DS100-KIT)	950g

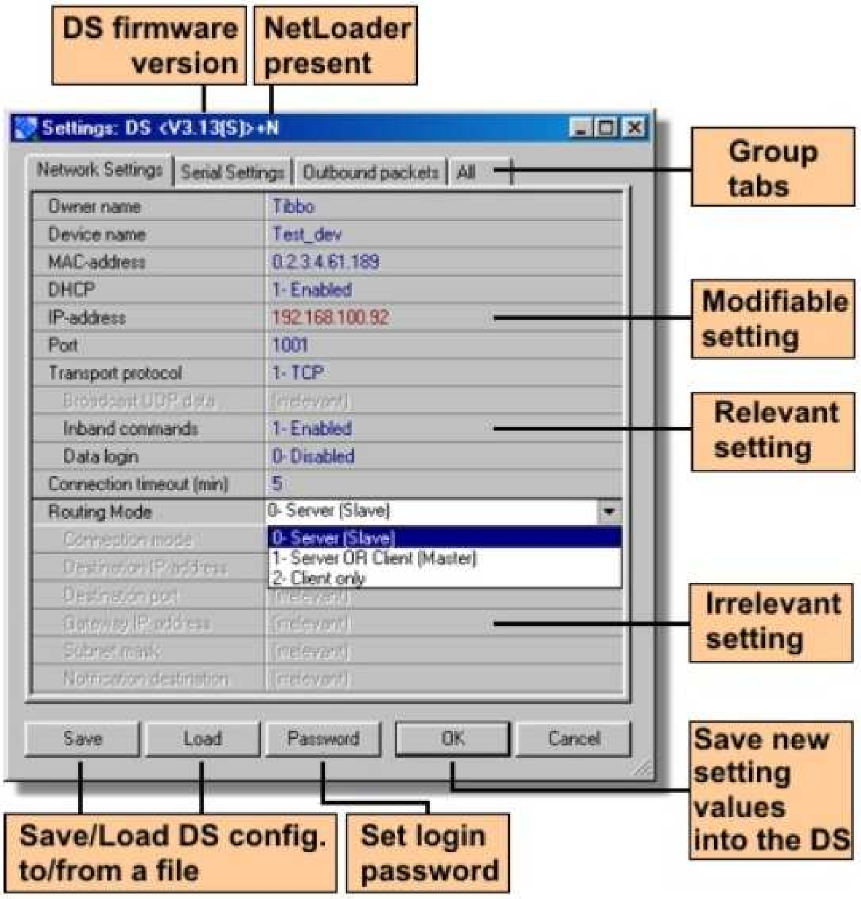
Operation setting parameters stored in the non-volatile memory (EEPROM) of the DS100 are functioning permanently. Once set by user, they remain intact even when the DS100 is powered off. Most settings require the DS100 to be rebooted for the new setting value to take effect. The baudrate (BR) setting defining the baudrate of the serial port could be quite an example.

As essential requirements for all of Ethernet devices, the DS100 has a unique MAC-address and an assigned valid IP-address to operate properly on the network. Since MAC-address is unique identification number preset into ROM during the production, it cannot be changed. In contrast, IP-address should be properly set according to the local area network setting. DHCP (Dynamic Host Configuration Protocol) is the popular and convenient way of setting IP-address automatically[8].

To route the data between attached serial device and the network host, the DS100 needs to establish and maintain a data connection. Depending on current transport protocol, the data connections can use TCP/IP or UDP/IP. In this project, UDP data connection has been used since considering the size of packets, a connection-less protocol is more effective than connection-based protocol.

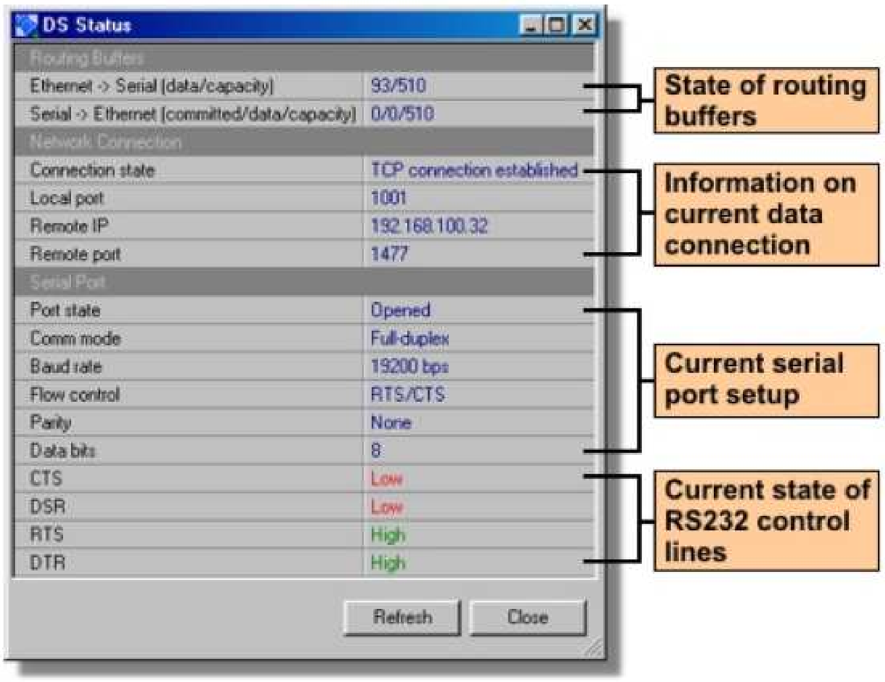
The DS100 can allow a single data connection only because the serial port is not a shared media and cannot be controlled by more than a single source at any given time. This is due to the COM port architecture of the PC - One peripheral device could be opened by one program at a time.

Figure 27. DS Setting Dialog[8]



Owner name	Tibbo
Device name	Test_dev
MAC-address	0.2.3.4.61.189
DHCP	1- Enabled
IP-address	192.168.100.92
Port	1001
Transport protocol	1- TCP
Broadcast UDP data	[irrelevant]
Inband commands	1- Enabled
Data login	0- Disabled
Connection timeout (min)	5
Routing Mode	0- Server (Slave)
Connection mode	0- Server (Slave) 1- Server OR Client (Master) 2- Client only
Destination IP-address	
Destination port	[irrelevant]
Gateway IP-address	[irrelevant]
Subnet mask	[irrelevant]
Notification destination	[irrelevant]

Figure 28. DS Routing Status Dialog[8]



DS Status

Routing Buffers	
Ethernet -> Serial (data/capacity)	93/510
Serial -> Ethernet (committed/data/capacity)	0/0/510
Network Connection	
Connection state	TCP connection established
Local port	1001
Remote IP	192.168.100.32
Remote port	1477
Serial Port	
Port state	Opened
Comm mode	Full-duplex
Baud rate	19200 bps
Flow control	RTS/CTS
Parity	None
Data bits	8
CTS	Low
DSR	Low
RTS	High
DTR	High

Refresh
Close
State of routing buffers
Information on current data connection
Current serial port setup
Current state of RS232 control lines

Listening port parameter defining the listening port number will be associated with transport protocols. Listening port usage of TCP/IP and UDP/IP is slightly different.

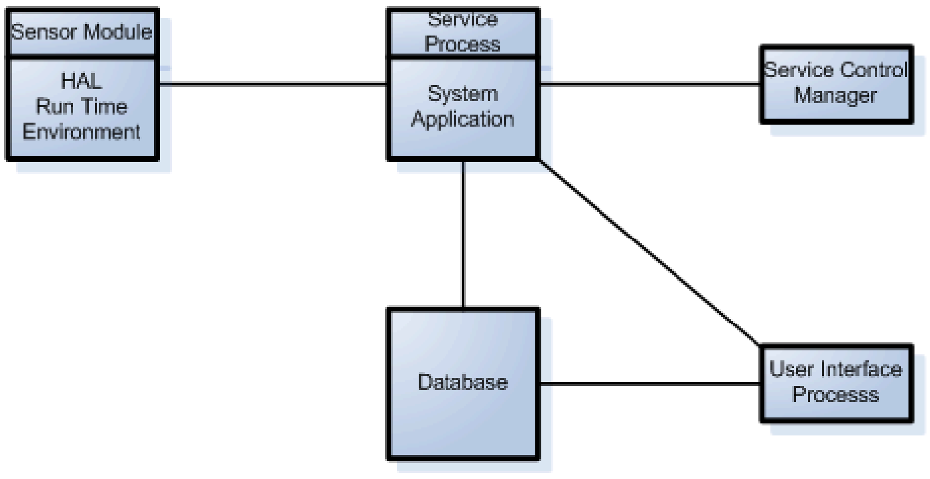
Looking at the UDP/IP transport protocol, there are two different working modes. First, when the routing mode is set as client, incoming connections are not allowed. Therefore, the listening port parameter is irrelevant. DS100 sends its own UDP datagram containing an automatically generated port number. Secondly, when the routing mode is server, the listening port parameter defines the listening port on which incoming UDP datagram is accepted and is also used as the port from which outgoing UDP datagram is sent. DS100 should be set as server in this project.

Chapter 3 Software Architecture

3.1 Software Architecture

In order to support experimentation and algorithm development, a flexible software environment is essential so that applications can be written in a high-level language while maintaining access to low-level hardware functions. The key wireless sensing node software functions are organized in the following layers:

Figure 29. Program Flow Diagram



Monitor/Hardware abstraction layer (HAL) provides routines for initialization, external communication, program loading and debugging, and interrupt processing. A packet protocol interpreter routes packets arriving from either the radio or the external RS-232 to internal tasks. Program loading can occur either through an attached device or through the radio.

Run-time environment, this real-time kernel on each node provides the low-level distributed wireless network infrastructure. The low-level controls for communication protocols as well as the sensor drivers are hosted at this level.

System Applications that perform signal processing computations and higher layer network functions (e.g., scheduling, routing). Written in a conventional high-level programming language such as C, new applications may be downloaded onto sensor nodes deployed in the field via the RF network.

A database management system can be an extremely complex set of software programs that controls the organization, storage and retrieval of data (fields, records and files) in a database. It also controls the security and integrity of the database. The DBMS accepts requests for data from the application program and instructs the operating system to transfer the appropriate data. When a DBMS is used, information systems can be changed much more easily as the organization’ s information requirements change. New categories of data can be added to the database without disruption to the existing system.

User interface applications hosted on PCs that allow users to perform various tasks and to interact with the sensor network. An interface for communicating with the network through a gateway is supported as well as display and logging of network information.

3.2 Service Applications

3.2.1 Window Service Communication Architecture

The Microsoft Windows operating system offers various facilities that make implementing the server-side portion of a client-server application easier. As you know, a server is a Windows application that performs server-side duties. Microsoft recommends that server applications be implemented as services.

A service is a normal Windows application containing additional infrastructure that enables it to receive special treatment by the operating system - for example, the ability to be remotely administered, allowing an administrator to start or stop the application from a remote machine. By turning your server application into a service, you'll get this and other features for free[11].

Following Table lists some of the services installed on my machine running Microsoft Windows 2000 and the name of the executable files containing the code for the services.

Table 6. Some Services Installed on Microsoft Windows 2000

Service Name	Description
Alerter	Notifies users and computers of administrative alerts
DHCP Client	Registers and updates IP addresses and DNS names
Event Log	Logs event messages issued by programs and Windows

Net Logon	Supports pass-through authentication of account logon events for computers in a domain
Plug and Play	Manages device installation and configuration, and notifies programs of device changes
Remote Procedure Call (RPC)	Provides the endpoint mapper and other miscellaneous RPC services
Task Scheduler	Enables a program to run at a designated time
Telephony	Provides Telephony API (TAPI) support
Windows Installer	Installs, repairs, and removes software according to instructions contained in .msi files
Windows Management Instrumentation	Provides system management information

First and foremost, a service application is just a 32-bit or 64-bit executable, so everything you already know about DLLs, structured exception handling, memory-mapped files, virtual memory, device I/O, thread-local storage, thread synchronization, Unicode, and other Windows facilities is available to a service. And this means that converting an existing server application into a service should be relatively easy and straightforward for you.

Second, you need to know that a service should have absolutely no user interface. Most services run on a server machine locked away in a closet somewhere. So if your service presented any user interface elements, such as message boxes, no user would be in front of the machine to see and then dismiss them. And, as you'll see later in this chapter, any windows created would probably appear on a window station or desktop different from the one the user was sitting in front of, and thus the message wouldn't be visible to the user anyway. Because a service won't have a user interface, it doesn't matter whether you choose to implement your service as a graphical user interface (GUI) application (with (w)WinMain as its entry point) or as a console user interface (CUI) application (with (w)main as its entry point).

If a service is not supposed to present any user interface, how do you configure the service? How can you start and stop a service? How can the service issue warnings or error messages? How can the service report statistical data about its performance? The answer to all these questions is that a service can be remotely administered. Windows offers a number of administrative tools that allow a service to be managed from other machines connected on the network so that it is not necessary for someone to physically check (or even have physical access to) the computer running the service. You are probably already familiar with many of these tools: the Microsoft Management Console (MMC), with its Services, Event Viewer, and System Monitor snap-ins; the registry editor; and the Net.exe command-line tool.

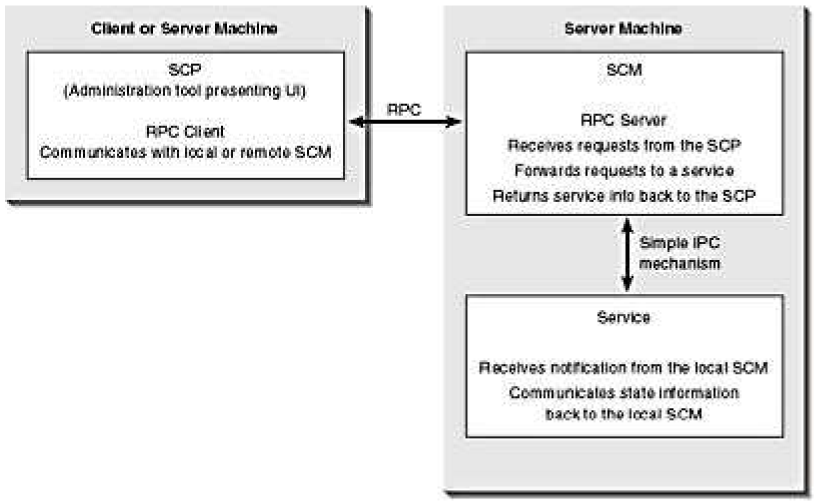
These facilities are provided by Windows to simplify the development effort of the service writer. They also give an administrator a consistent way to manage machines remotely and locally. Note that these facilities are not exclusive to services: any application (or device driver) can take advantage of them.

Three types of components are involved in making services work[11]:

- Service Control Manager (SCM, pronounced scum) - Each Windows 2000 system ships with a component called the Service Control Manager. This component lives in the Services.exe file; it is automatically invoked when the operating system boots, and terminates when the system is shut down. The SCM runs with system privileges and provides a unified and secure means of controlling service applications. The SCM is responsible for communicating with the various services, telling them to start, stop, pause, continue, and so on
- Service application - This is an application that usually presents a user interface that allows a user to start, stop, pause, continue, and otherwise control all the services installed on a machine. The service control program calls special Windows functions that let it talk to the SCM.
- Service Control Program (SCP) - This is an application that usually presents a user interface that allows a user to start, stop, pause, continue, and otherwise control all the services installed on a machine. The service control program calls special Windows functions that let it talk to the SCM.

Figure 30 shows how all these components communicate with one another. Notice that SCP applications do not communicate with services directly; all communication goes through the SCM. This architecture is precisely what makes the remote administration transparent to the SCP and service applications. It is possible to implement an architecture and a protocol that enables your SCP application to talk directly with your service application, but you must write the communication code yourself.

Figure 30. Windows Service Communication Architecture[11]


Client or Server Machine
SCP
(Administration tool presenting UI)
RPC Client
Communicates with local or remote SCM
RPC
Server Machine
SCM
RPC Server
Receives requests from the SCP
Forwards requests to a service
Returns service info back to the SCP
Simple IPC mechanism
Service
Receives notification from the local SCM
Communicates state information back to the local SCM

Of these three components, you will never implement the SCM itself. Microsoft implements the SCM and packages it into every version of Windows 2000. What you will implement are services and SCPs. This chapter will cover what you need to know to design and implement a service, and the next chapter will cover the details of writing an SCP.

3.2.2 Services Snap-In

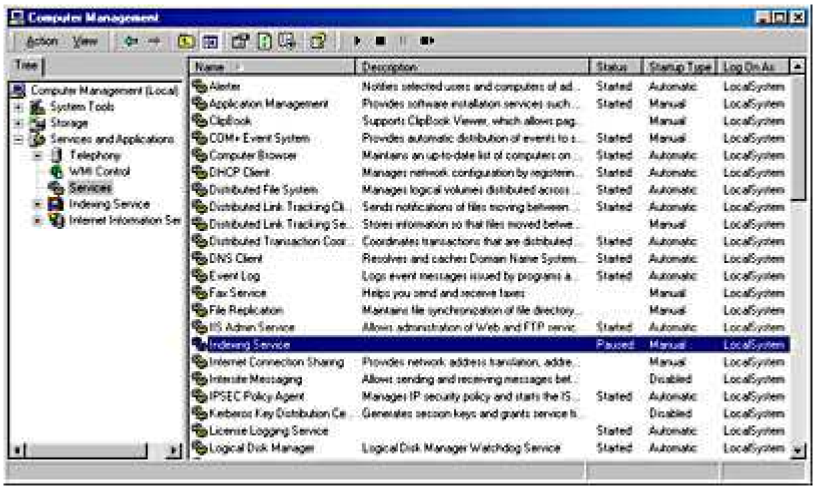
The SCP application with which you should become most familiar is the Services snap-in, shown in Figure 31. This snap-in shows the list of all services installed on the target machine. The Name and Description columns identify each service's name and offer an informative description of the service's function. The Status column indicates whether the service is Started, Paused, or Stopped (blank entries indicate "stopped"). The Startup Type column indicates when the SCM should invoke the service, and the Log On As column indicates the security context used by the service when it is running.

This information is kept in the SCM's database, which lives inside the registry under the following subkey:

HKEY_LOCAL_MACHINE₩SYSTEM₩CurrentControlSet₩Services

You should never access this subkey directly; instead, an SCP should call the Windows functions (discussed in the next chapter) that manipulate the database in this subkey. Directly modifying the contents of this key will yield unpredictable results. When you install a product that includes a service, the setup program for that product is an SCP that adds the service's information to the SCM's database.

Figure 31. Services Snap-in[11]



Name	Description	Status	Startup Type	Log On As
Alerter	Notifies selected users and computers of ad...	Started	Automatic	LocalSystem
Application Management	Provides software installation services such...	Started	Manual	LocalSystem
ClipBook	Supports ClipBook Viewer, which allows pag...		Manual	LocalSystem
COM+ Event System	Provides automatic distribution of events to s...	Started	Manual	LocalSystem
Computer Browser	Maintains an up-to-date list of computers on...	Started	Automatic	LocalSystem
DHCP Client	Manages network configuration by registerin...	Started	Automatic	LocalSystem
Distributed File System	Manages logical volumes distributed across...	Started	Automatic	LocalSystem
Distributed Link Tracking Cli...	Sends notifications of files moving between...	Started	Automatic	LocalSystem
Distributed Link Tracking Se...	Stores information so that files moved betwe...		Manual	LocalSystem
Distributed Transaction Coor...	Coordinates transactions that are distributed...	Started	Automatic	LocalSystem
DNS Client	Resolves and caches Domain Name System...	Started	Automatic	LocalSystem
Event Log	Logs event messages issued by programs a...	Started	Automatic	LocalSystem
Fax Service	Helps you send and receive faxes		Manual	LocalSystem
File Replication	Maintains file synchronization of file directory...		Manual	LocalSystem
IIS Admin Service	Allows administration of Web and FTP servic...	Started	Automatic	LocalSystem
Indexing Service		Paused	Manual	LocalSystem
Internet Connection Sharing	Provides network address translation, addre...		Manual	LocalSystem
Intersite Messaging	Allows sending and receiving messages bet...		Disabled	LocalSystem
IPSEC Policy Agent	Manages IP security policy and starts the IS...	Started	Automatic	LocalSystem
Kerberos Key Distribution Ce...	Generates session keys and grants service ti...		Disabled	LocalSystem
License Logging Service		Started	Automatic	LocalSystem
Logical Disk Manager	Logical Disk Manager Watchdog Service	Started	Automatic	LocalSystem

You can view a remote SCM's service database by selecting the Computer Management node in the left pane of the Computer Management console and then choosing Connect To Another Computer from the Action menu.

So now that you're looking at the Services snap-in, you're probably wondering about all the tasks you can perform with it. Here are the most common operations[11]:

Start a service

The administrator starts a service by selecting the service from the list and clicking the Start toolbar button. Only services with a Startup Type of Automatic or Manual can be started; disabled services cannot. Disabling a service is useful for troubleshooting problems with a machine.

Stop a service

The administrator stops a service by selecting the service and clicking the Stop toolbar button. Note that some services do not allow themselves to be stopped after they are started. The Event Log service is an example; it stops only when the machine shuts down.

Pause and resume a service

The administrator pauses a service by selecting a running service and clicking the Pause toolbar button. Note that most services do not allow themselves to be paused. Also note that "pause" has no exact definition. For one service, pausing can mean that the service won't accept client requests until it finishes processing the outstanding requests. For another service, pausing can mean that the service can no longer process any of its operations. Paused services can be resumed by clicking the Start toolbar button.

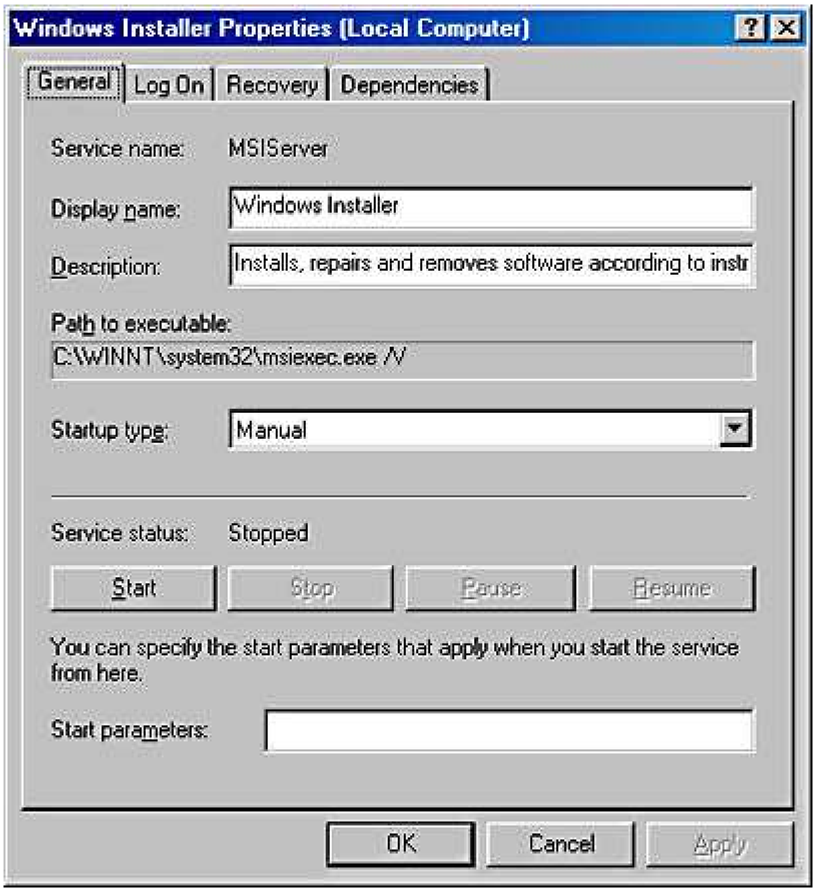
Restart a service

The administrator restarts a service by selecting a running or paused service and clicking the Restart toolbar button. Restarting a service causes the snap-in to stop the service and then start the service. This is simply a convenience feature and is very useful when debugging your own service.

The preceding list certainly accounts for 99 percent of what administrators do with the Services snap-in, but the snap-in can also be used to reconfigure a service. To change a service's settings, you select the service and then display its Properties dialog box. This dialog box contains four tabs; each tab allows the administrator to reconfigure parts of the selected service. The configurable settings are discussed in the following sections.

The General tab (shown in Figure. 32) allows the administrator to examine and reconfigure general information about the service. The first fact you need to know is that each service goes by two string names: an internal name (used for programmatic purposes) and a display name (a pretty string presented to administrators and users). After being added to the machine's service database, a service's internal name cannot be altered, but the administrator can modify the service's display name and description. The General tab also shows the service's pathname but does not allow the administrator to change it. (This is a limitation imposed by the tab, not by the system.) The administrator can change the service's Startup Type to one of the following[11]:

Figure 32. General Tap for Windows Installer Service[11]


Windows Installer Properties (Local Computer)

Service name: MSIServer

Display name: Windows Installer

Description: Installs, repairs and removes software according to instr

Path to executable:
C:\WINNT\system32\msiexec.exe /V

Startup type: Manual

Service status: Stopped

You can specify the start parameters that apply when you start the service from here.

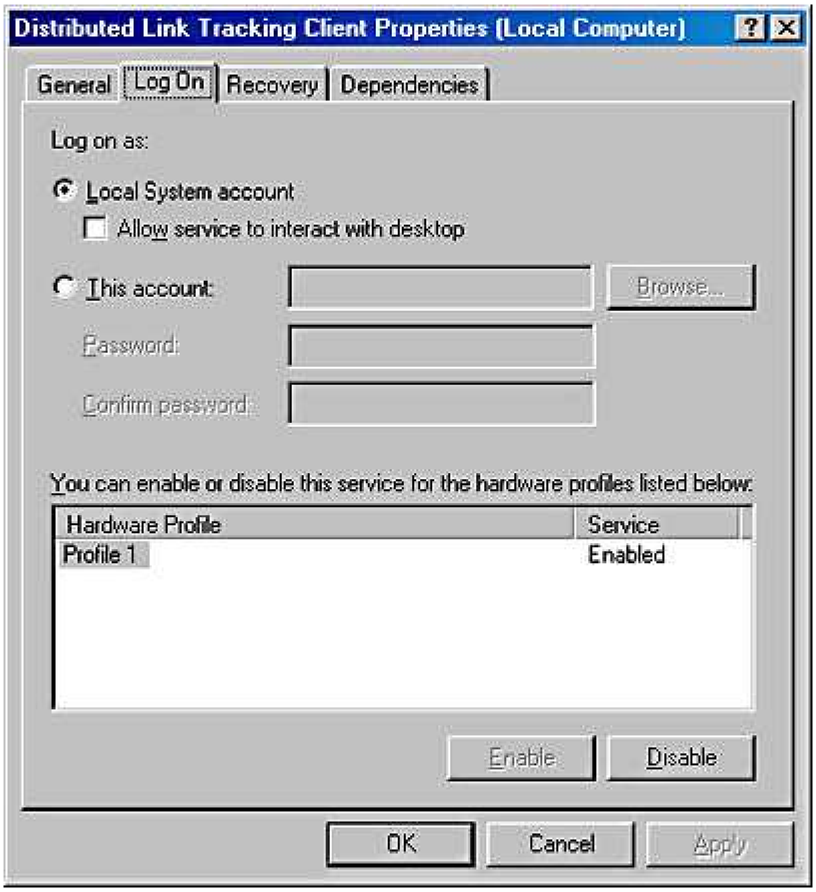
Start parameters:

- Automatic - One of the features of a service is that the SCM can automatically start it for you. If the service has a Startup Type of Automatic, the SCM spawns the service when the operating system boots. It is important to note that automatic services run before any user interactively logs on to the machine. In fact, many machines that run Windows are set up to run only services - no one ever logs on to the machine interactively. For example, machines running Windows and the

Server service allow clients to access subdirectories, files, and printers on a networked machine.

- Manual - A manual service tells the SCM not to start the service when the machine boots. An administrator can start this service manually using an SCP. A manual service (alternately known as a demand-start service) will also start when another service that depends on the manual service is started. I'll talk about service dependencies more in the next chapter.
- Disabled - A disabled service tells the SCM not to start it under any circumstance. You disable the DHCP Client service when you manually assign an IP address to your machine rather than have it dynamically obtain an IP address from a machine running the DHCP Server service. Disabling a service is also quite useful when troubleshooting a system by allowing you to take a specific service out of the equation.

Figure 33. Log On Tab for the Distributed Link Tracking Service[11]


Distributed Link Tracking Client Properties (Local Computer)

Log on as:

(● Local System account) [✓]

☐ Allow service to interact with desktop

○ This account:

Password:

Confirm password:

You can enable or disable this service for the hardware profiles listed below:

Hardware Profile	Service
Profile 1	Enabled

In addition to configuring the actual service, the administrator can reconfigure the security context under which the service will execute on the Log On tab, shown in Figure 33. The security context can be one of the following[11]:

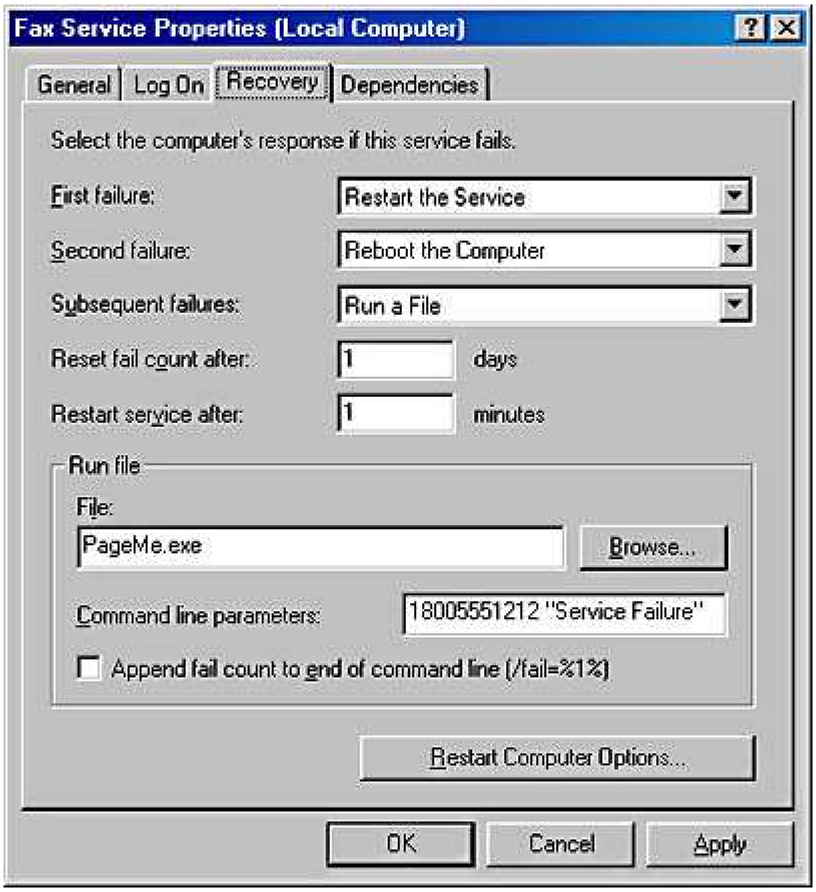
- LocalSystem Account – A service running under the LocalSystem account can do just about anything on the computer: open any file, shut down the machine, change the system time, and so on. A service running under the LocalSystem account can optionally be allowed to interact with the desktop. Most services don't require this option, and you are strongly discouraged from using it.

- This Account - A service can also execute under a specific security context (identified by a user's name and password). This restricts the service to accessing the resources accessible to the specified account.

The Log On tab also allows the administrator to specify which hardware profiles the service is enabled in. Hardware profiles allow you to configure services according to your hardware configuration. For example, you might want the fax service to run when your laptop computer is docked and not run when it is undocked.

The Recovery tab, shown in Figure 34, allows the administrator to tell the SCM what actions to perform should the service terminate abnormally. Abnormal termination means that the service stopped without reporting a status of SERVICE_STOPPED (discussed later in this chapter). For the first, second, and subsequent attempts, the SCM can do nothing, automatically restart the service, run an executable, or reboot the computer. Note that running an executable and rebooting the computer can fail if the account under which the service is running doesn't have the appropriate privileges or permissions.

Figure 34. Recovery tab for the Fax Service Service[11]


Fax Service Properties (Local Computer)

General | Log On | Recovery | Dependencies

Select the computer's response if this service fails.

First failure: Restart the Service

Second failure: Reboot the Computer

Subsequent failures: Run a File

Reset fail count after: 1 days

Restart service after: 1 minutes

Run file

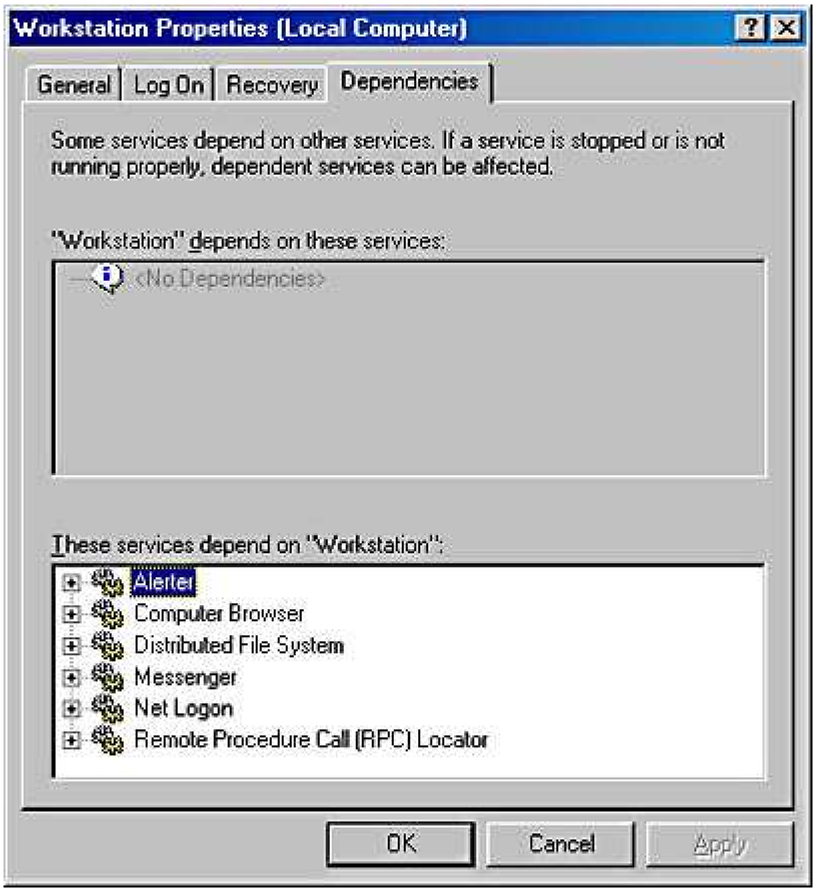
File: PageMe.exe

Command line parameters: 18005551212 "Service Failure"

☐ Append fail count to end of command line (/fail=%1%)

The Dependencies tab, shown in Figure 35, shows the services on which the selected service depends and also what services depend on the selected service. In the figure, you'll see six services dependent on the Workstation service. If the administrator attempts to stop the Workstation service and any dependent services are running, the SCM fails the call. Many SCP programs are written to notify the user that dependent services are running, and to allow the user to choose whether to also stop the dependent services. The Dependencies tab does not allow an administrator to modify any of these dependencies.

Figure 35. Dependencies tab for the Workstation service[11]


Workstation Properties (Local Computer)

Some services depend on other services. If a service is stopped or is not running properly, dependent services can be affected.

"Workstation" depends on these services:

<No Dependencies>

These services depend on "Workstation":

Alerter
Computer Browser
Distributed File System
Messenger
Net Logon
Remote Procedure Call (RPC) Locator

3.2.3 The Windows Service Application Architecture

In this section, the additional infrastructure that turns a server application into a service will be explained, thus allowing your application to be remotely administered. Microsoft's service architecture seems to be a little difficult to understand at first. The difficulty is due to the fact that every service process always contains at least two threads, and these threads must communicate with one another. So you have to deal with thread synchronization issues and inter-thread communication issues.

Another issue that should be considered is that a single executable file can contain several services. Many services can be contained inside the Services.exe file. Most of these services (such as DHCP Client, Messenger, and Alerter) are fairly simple in their implementation. It would be very inefficient if each of these services had to run as a separate process, with its own address space and additional process overhead. Because of this overhead, Microsoft allows a single executable to contain several services. The Services.exe file actually contains about 20 different services inside of it, including the three just mentioned.

When designing a service executable, you must concern yourself with three kinds of functions[11]:

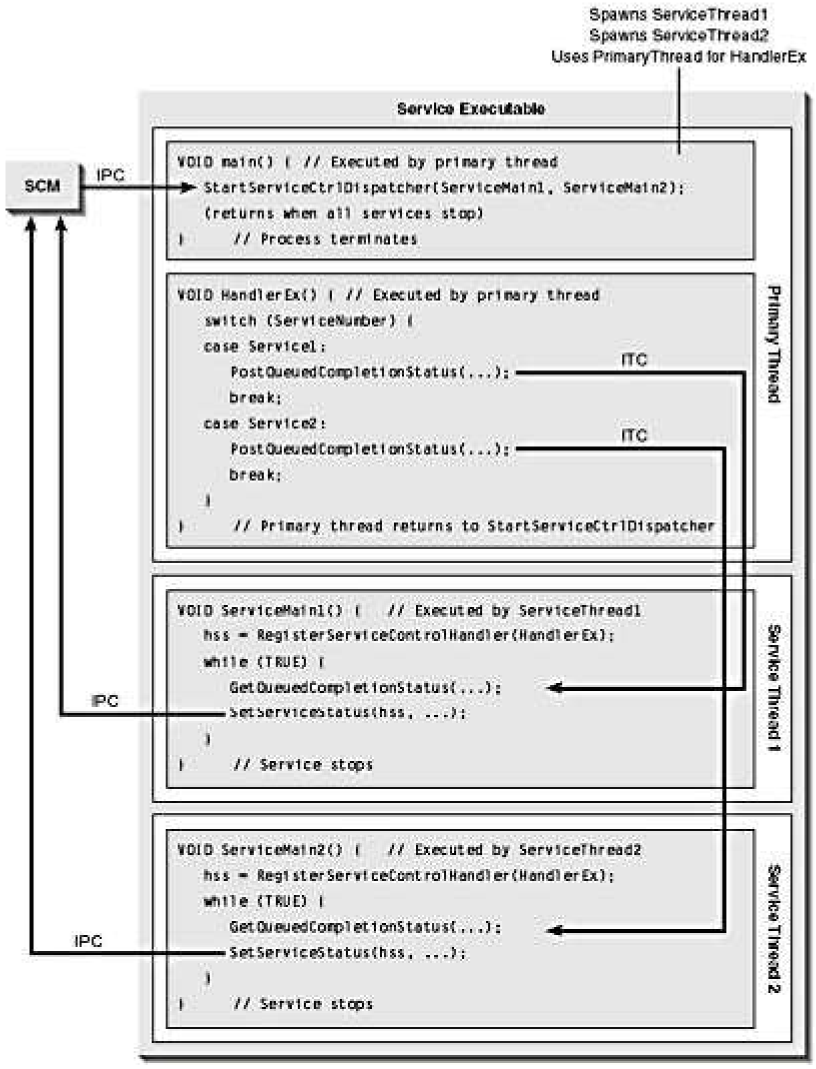
Process's entry-point function This function is your standard (w)main or (w)WinMain function with which you should be extremely familiar by now. For a service, this function initializes the process as a whole and then calls a special function that connects the process with the local system's SCM. At this point, the SCM takes control of your primary thread for its own purposes. Your code will regain control only when all of the services in the executable have stopped running.

Service's ServiceMain function You must implement a ServiceMain function for each service contained inside your executable file. To run a service, the SCM spawns a thread in your process that executes your ServiceMain function. When the thread returns from ServiceMain, the SCM considers the service stopped. Note that this function does not have to be called ServiceMain; you can give it any name you desire.

Service's HandlerEx function Each service must have a HandlerEx function associated with it. The SCM communicates with the service by calling the service's HandlerEx function. The code in the HandlerEx function is executed by your process's primary thread. The HandlerEx function either executes the necessary action, or it must communicate the SCM's instructions to the thread that is executing the service's ServiceMain function by using some form of interthread communication. Note that each service can have its own HandlerEx function, or multiple services (in a single executable) can share a single HandlerEx function. One of the parameters passed to the HandlerEx function indicates which service the SCM wishes to communicate with. Note that this function does not have to be called HandlerEx; you can give it any name you desire.

Figure 36 will help you put this architecture in perspective. It shows the functions necessary to implement a service executable that houses two services as well as the lines of interprocess communication (IPC) and interthread communication (ITC). In the upcoming sections, I will examine these three functions in detail and flesh out exactly what their responsibilities are. I recommend that you refer to this figure while reading.

Figure 36. Windows Service Application Architecture[11]



```
VOID main() { // Executed by primary thread
   StartServiceCtrlDispatcher(ServiceMain1, ServiceMain2);
   (returns when all services stop)
}     // Process terminates
```

```
VOID HandlerEx() { // Executed by primary thread
   switch (ServiceNumber) {
   case Service1:
      PostQueuedCompletionStatus(...);
      break;
   case Service2:
      PostQueuedCompletionStatus(...);
      break;
   }
}     // Primary thread returns to StartServiceCtrlDispatcher
```

```
VOID ServiceMain1() {   // Executed by ServiceThread1
   hss = RegisterServiceControlHandler(HandlerEx);
   while (TRUE) {
      GetQueuedCompletionStatus(...);
      SetServiceStatus(hss, ...);
   }
}     // Service stops
```

```
VOID ServiceMain2() {   // Executed by ServiceThread2
   hss = RegisterServiceControlHandler(HandlerEx);
   while (TRUE) {
      GetQueuedCompletionStatus(...);
      SetServiceStatus(hss, ...);
   }
}     // Service stops
```

Following are our implementations and explanations.

List 2. SNServicedLib.cs

```
using System;
using System.IO;
using System.Text;
using System.Threading;
using JSSHandlingDatabase;

namespace SNServiceLib
{
  /// <summary>
  /// DLL Implementation of Service
  /// </summary>
  public class SNServiceLibClass
  {
    private Thread svThread;
    private StreamWriter write;
    private string logFilePath = @"C:₩ServiceLog.txt";
    private string curTime;

    public SNServiceLibClass()
    {
      //
      // Constructor
      //
      svThread = new Thread(new ThreadStart(this.ServiceLoop));
      write = new StreamWriter(logFilePath, true, Encoding.Unicode);
    }

    protected void ServiceLoop()
    {
      while (true)
      {
        HandlingDatabase db = new HandlingDatabase();
        db.InsertData();
        curTime = System.DateTime.Now.ToString();
        write.WriteLine(curTime + " : Running Service");
        Thread.Sleep(10000);
      }
    }

    public void Start()
    {
      svThread.Start();
      curTime = System.DateTime.Now.ToString();
      write.WriteLine(curTime + " : Start Service");
    }

    public void Stop()
    {
      svThread.Abort();
      curTime = System.DateTime.Now.ToString();
      write.WriteLine(curTime + " : Stop Service");
      write.Close();
    }

    public void Suspend()
    {
      svThread.Suspend();
      curTime = System.DateTime.Now.ToString();
      write.WriteLine(curTime + " : Pause Service");
    }

    public void Resume()
    {
      svThread.Resume();
      curTime = System.DateTime.Now.ToString();
      write.WriteLine(curTime + " : Continue Service");
    }
  }
}
```

List 3. SNService.cs

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.ServiceProcess;
using SNServiceLib;

namespace SNService
{
  public class MyService : System.ServiceProcess.ServiceBase
  {
    /// <summary>
    /// Essential Designer Variables
    /// </summary>
    private System.ComponentModel.Container components = null;
    private SNServiceLibClass service;

    public SNService()
    {
      // Windows.Forms needs this
      InitializeComponent();
      // TODO: Define After InitComponent Jobs Here
    }

    // Main Entrance of Process
    static void Main()
    {
```

```
  System.ServiceProcess.ServiceBase[] ServicesToRun;

  // More than two can be run inside one process
  // If you want insert another services,
  // change next lines. For exmaple
  //
  // ServicesToRun = New System.ServiceProcess.ServiceBase[]
  // {new Service1(), new MySecondUserService()};
  //
  ServicesToRun = new System.ServiceProcess.ServiceBase[]
  {
    new SNService()
  };

  System.ServiceProcess.ServiceBase.Run(ServicesToRun);
}

/// <summary>
/// Methods for Designer
/// Do not edit this code by code editor
/// </summary>
private void InitializeComponent()
{
  //
  // SNService
  //
```

```
    this.CanPauseAndContinue = true;
    this.ServiceName = "SNService";
}

/// <summary>
/// Close all resources
/// </summary>
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if (components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}

/// <summary>
/// Define what services do here
/// </summary>
protected override void OnStart(string[] args)
{
    // TODO: Service starting code here
    service = new MyServiceLibClass();
```

```
            service.Start();
        }

        /// <summary>
        /// Stop Service
        /// </summary>
        protected override void OnStop()
        {
            // TODO: What should do before stop service
            service.Stop();
        }

        protected override void OnPause()
        {
            service.Suspend();
        }

        protected override void OnContinue()
        {
            service.Resume();
        }

    }
}
```

List 4. ProjecInstaller.cs

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Configuration.Install;

namespace SNService
{
  /// <summary>
  /// Summary on ProjectInstaller
  /// </summary>
  [RunInstaller(true)]
  public class ProjectInstaller :System.Configuration.Install.Installer
  {
    private System.ServiceProcess.ServiceProcessInstaller
      serviceProcessInstaller1;
    private System.ServiceProcess.ServiceInstaller serviceInstaller1;
    /// <summary>
    /// Essential Designer Variables
    /// </summary>
    private System.ComponentModel.Container components = null;

    public ProjectInstaller()
    {
      // This call is for designer
  InitializeComponent();
}

/// <summary>
/// Close all resources
/// </summary>
protected override void Dispose( bool disposing )
{
  if( disposing )
  {
    if(components != null)
    {
      components.Dispose();
    }
  }
  base.Dispose( disposing );
}


#region Component Designer Generated Code
/// <summary>
/// Methods for Designer
/// Do not edit this code by code editor
/// </summary>
private void InitializeComponent()
{
```

```
this.serviceProcessInstaller1 = new
  System.ServiceProcess.ServiceProcessInstaller();
this.serviceInstaller1 = new System.ServiceProcess.ServiceInstaller();
//
// serviceProcessInstaller1
//
this.serviceProcessInstaller1.Account =
  System.ServiceProcess.ServiceAccount.LocalSystem;
this.serviceProcessInstaller1.Password = null;
this.serviceProcessInstaller1.Username = null;
this.serviceProcessInstaller1.AfterInstall += new
  System.Configuration.Install.InstallEventHandler(
  this.serviceProcessInstaller1_AfterInstall);
//
// serviceInstaller1
//
this.serviceInstaller1.DisplayName = "SNService";
this.serviceInstaller1.ServiceName = "SNService";
//
// ProjectInstaller
//
this.Installers.AddRange(new System.Configuration.Install.Installer[]
  {
    this.serviceProcessInstaller1, this.serviceInstaller1
  }
);
```

```
        }
        #endregion

        private void serviceProcessInstaller1_AfterInstall(object sender,
          InstallEventArgs e)
        {

        }

        private void serviceInstaller1_AfterInstall(
          object sender, InstallEventArgs e)
        {

        }

    }
}
```

3.3 User Interface

User interface applications hosted on personal computer that allow users to perform various tasks and to interact with the sensor network. An interface for communicating with the network through gateway is supported as well as display and logging of network information.

3.3.1 User Datagram Protocol

There are two protocols at the Transport Layer that Application Layer protocols typically use for transporting data: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). UDP is the Transport Layer protocol that offers a minimum of services, but also has the minimum overhead for Application Layer protocols that do not require an end-to-end reliable delivery service.

UDP is a minimal Transport Layer protocol that is a direct reflection of the datagram services of IP, except that UDP provides a method to pass the message portion of the UDP message to the Application Layer protocol. UDP has the following characteristics[12]:

- Connectionless - UDP messages are sent without a UDP-based connection establishment negotiation.
- Unreliable - UDP messages are sent as datagrams without sequencing or acknowledgment. The Application Layer protocol must recover lost messages. Typical UDP-based Application Layer protocols either provide their own reliable service or retransmit UDP messages periodically or

after a defined time-out value.

- Provides identification of Application Layer protocols - UDP provides a mechanism to send messages to a specific Application Layer protocol or process on an internetwork host. The UDP header provides both source and destination process identification.
- Provides checksum of UDP message - The UDP header provides a 16-bit checksum over the entire UDP message.

UDP does not provide the following services for end-to-end delivery:[12]

- Buffering - UDP does not provide any buffering of incoming or outgoing data. The Application Layer protocol must provide all buffering.
- Segmentation - UDP does not provide any segmentation of large blocks of data. Therefore, the application must send data in small enough blocks so that the IP datagrams for the UDP messages are no larger than the Maximum Transmission Unit (MTU) of the interface on which they are sent; otherwise IP on the sending host fragments the UDP message.
- Flow control - UDP does not provide any sender-side or receiver-side flow control. UDP message senders can react to the receipt of an Internet Control Message Protocol (ICMP) Source Quench message, but it is not required.

Because UDP does not provide any services beyond Application Layer protocol identification and a checksum, it is hard to imagine why UDP is needed at all. However, the following are specific uses for sending data using UDP:[12]

l Lightweight protocol

To conserve memory and processor resources, some Application Layer protocols require the use of a lightweight protocol that performs a specific function using a simple exchange of messages. A good example of a lightweight protocol is Domain Name System (DNS) name queries. Typically, a DNS client sends a DNS Name Query message to a DNS server. The DNS server responds with a DNS Name Query Response message. If the DNS server does not respond, the DNS client retransmits the DNS Name Query.

Imagine the resources required at the DNS server if all the DNS clients used TCP rather than UDP. All DNS interactions would be sent reliably, but the DNS server would have to support hundreds or, on the Internet, thousands of TCP connections. The low-overhead solution of using UDP is the best choice for simple request-reply-based Application Layer protocols.

l Reliability provided by the Application Layer protocol

If the Application Layer protocol provides its own reliable data transfer service, there is no need for the reliable services of TCP. Examples of reliable Application Layer protocols are Trivial File Transfer Protocol (TFTP) and Network File System (NFS).

l Reliability not required due to periodic advertisement process

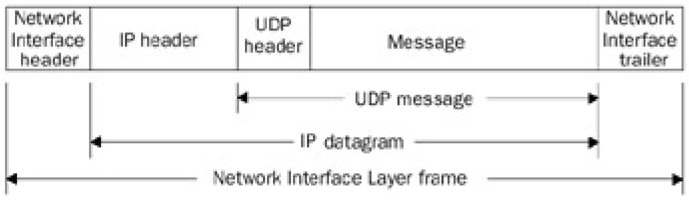
If theApplication Layer protocol periodically advertises information, reliable delivery is not required. If an advertisement is lost, it is announced again at the period interval. An example of an Application Layer protocol that uses periodic advertisements is the Routing Information Protocol (RIP).

l One-to-many delivery

UDP is used as the Transport Layer protocol whenever Application Layer data must be sent to multiple destinations using an IP multicast or broadcast address. TCP can be used only for one-to-one delivery. For example, a broadcast NetBIOS Name Query is sent using UDP.

UDP messages are sent as IP datagrams. A UDP message consisting of a UDP header and a message is encapsulated with an IP header using IP Protocol number 17 (0x11). The message can be a maximum size of 65,507 bytes: 65,535 minus the minimum-size IP header (20 bytes) and the UDP header (8 bytes). The resulting IP datagram is thenencapsulated with the appropriate Network Interface Layer header and trailer. Figure 37 shows the resulting frame. UDP is described in RFC 768[12].

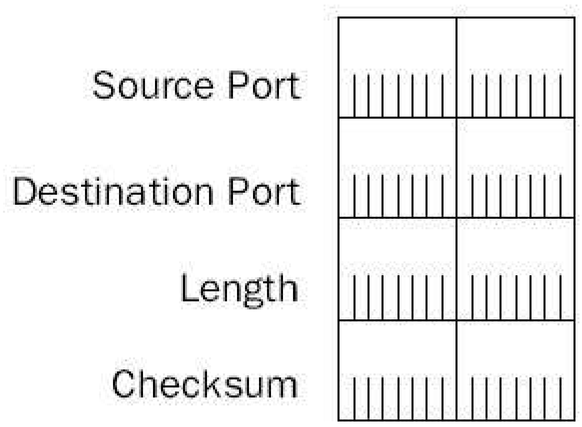
Figure 37. UDP Message Encapsulation[12]



In the IP header of UDP messages, the Source IP Address field is set to the host interface that sent the UDP message. The Destination IP Address field is set to the unicast address of a specific host, an IP broadcast address, or an IP multicast address.

The UDP header is a fixed-length size of 8 bytes consisting of four fixed-length fields, as Figure 38 shows.

Figure 38. The Structure of the UDP Header[12]



The fields in the UDP header are defined as follows[12]:

l Source Port

A 2-byte field used to identify the source Application Layer protocol sending the UDP message. The use of a source port is optional and, when not used, is set to 0. IP multicast traffic, such as videocasts sent using UDP, can use 0 because no reply to the video traffic is assumed. Typical Application Layer protocols use the source port of the incoming UDP message as the destination port for replies.

l Destination Port

A 2-byte field used to identify the destination Application Layer protocol. The combination of the IP header's destination IP address and the UDP header's destination port provides a unique, globally significant address for the process to which the message is sent.

l Length

A 2-byte field used to indicate the length in bytes of the UDP message (UDP header and message). The minimum length is 8 bytes (the UDP header's size), and the maximum is 65,515 bytes (maximum-sized IP datagram of 65,535 bytes minus the minimum-sized IP header of 20 bytes). The actual maximum length is confined by the MTU of the link on which the UDP message is sent. The Length field is a redundant field. The UDP length can always be calculated from the Total Length and the IP Header Length fields in the IP header (UDP length = payload length = total length - 4×IP header length [in 32-bit words]).

l Checksum

A 2-byte field that provides a bit-level integrity check for the UDP message (UDP header and message). The UDP checksum calculation uses the same method as the IP header checksum over the UDP pseudo header, the UDP header, the message, and, if needed, a padding byte of 0x00. The padding byte is used only if the message's length is an odd number of bytes. The use of the UDP Checksum field is optional. If not used, the UDP Checksum field is set to 0.

A UDP port defines a location or message queue for the delivery of messages for Application Layer protocols using UDP services. Included in each UDP message is the source port (the message queue from which the message was sent) and a destination port (the message queue to which the message was sent). The Internet Assigned Numbers Authority (IANA) assigns port numbers, known as well-known port numbers, to specific Application Layer protocols. Table 7 shows well-known UDP port numbers used by the Windows Server 2003 family and Windows XP components.

Table 7. Well-Known UDP Port Numbers[12]

Port Number	Application Layer Protocol
53	DNS
67	BOOTP client (Dynamic Host Configuration Protocol [DHCP])
68	BOOTP server (DHCP)
69	TFTP
137	NetBIOS Name Service
138	NetBIOS Datagram Service
161	Simple Network Management Protocol (SNMP)
445	Direct hosting of Server Message Block (SMB) datagrams over

	TCP/IP
520	RIP
1812, 1813	Remote Authentication Dial-In User Service (RADIUS)

UDP provides a connectionless and unreliable delivery service for applications that do not require the guaranteed delivery service of TCP. Application Layer protocols use UDP for lightweight interaction, for broadcast or multicast traffic, or when the Application Layer protocol provides its own reliable delivery service. The UDP header provides a checksum and the identification of source and destination port numbers to multiplex UDP message data to the proper Application Layer protocol. To consider the characteristics of sensor network traffic, UDP can be perfectly fit in.

3.3.2 ODBC

Open DataBase Connectivity (ODBC) is an Application Programming Interface (API) that allows a programmer to abstract a program from a database. When writing code to interact with a database, you usually have to add code that talks to a particular database using a proprietary language. If you want your program to talk to an Access, Fox and Oracle databases you have to code your program with three different database languages.

When programming to interact with ODBC you only need to talk the ODBC language (a combination of ODBC API function calls and the SQL language). The ODBC Manager will figure out how to contend with the type of database you are targeting. Regardless of the database type you are using, all of your calls will be to the ODBC API. All that you need to do is having installed an ODBC driver that is specific to the type of database you will be using.

Over the years Microsoft has modified what they call ODBC. It used to be that you would download their ODBC manager. You would then download and install any database specific ODBC drivers that you need. However times have changed and so has marketing.

Now there are a bunch of different layers of database software. You can find ADO, RDO, OLE DB and a variety of others. Microsoft has merged all of these technologies into one nifty installation package called MDAC (Microsoft Data Access Components). Basically it consists of several components that provide various database technologies; including ODBC. MDAC is a royalty-free redistributable package that you can install on a Windows machine without a cost.

Here are some examples of using Win32::ODBC used as a CGI script. It fully outlines ODBC usage basics.

1. First you need to create a DSN (Data Source Name) which is a name that represents the database file (or connection) and ODBC driver as well as user id and password.

2. Second you add the following USE line to the beginning of your Perl script: use Win32::ODBC;

3. Third you open a connection to your database with (note that this

example checks for failure):

```
$DSN = "My DSN";
if (!($db = new Win32::ODBC($DSN))){
    print "Error connecting to $DSN₩n";
    print "Error: " . Win32::ODBC::Error() . "₩n";
    exit;
}
```

4. Fourth you execute your SQL command (NOTE: due to backward compatibility with NT::ODBC the Sql() method returns undef if it was successful and a non zero integer error number if it fails):

```
$SqlStatement = "SELECT * FROM Foo";
if ($db->Sql($SqlStatement)) {
  print "SQL failed.₩n";
  print "Error: " . $db->Error() . "₩n";
  $db->Close();
  exit;
}
```

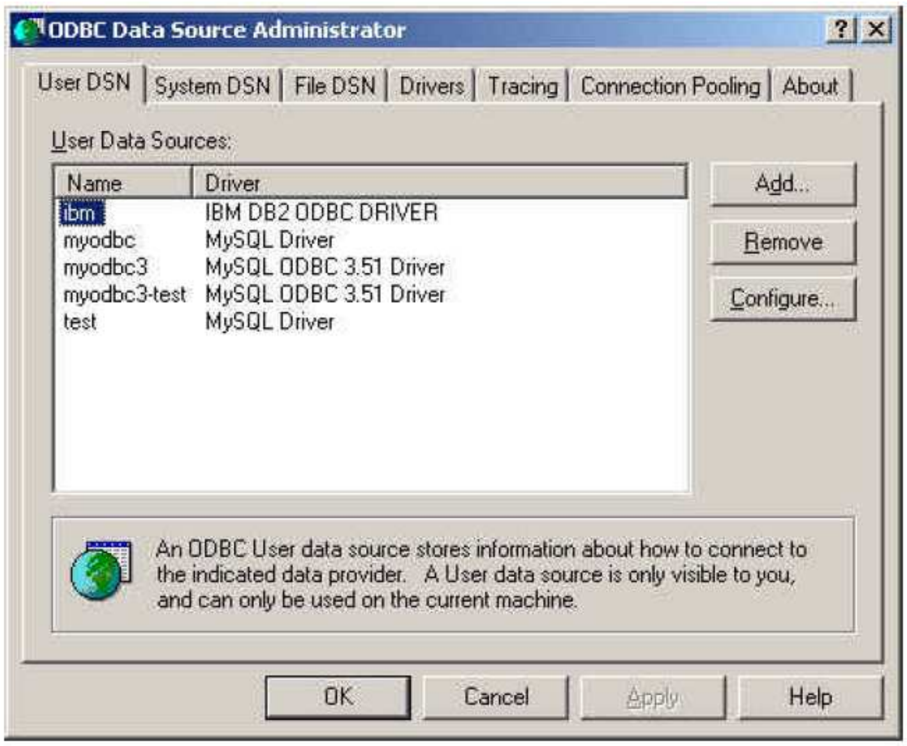
5. Fifth you fetch the next row of data until there are no more left to fetch. For each row you retrieve data and process it:

```
while($db->FetchRow()){
  undef %Data;
  %Data = $db->DataHash();
  ...process the data...
}
```

6. Sixth you close the connection to the database:

```
$db->Close();
```

Figure 39. ODBC Data Source Administrator[11]

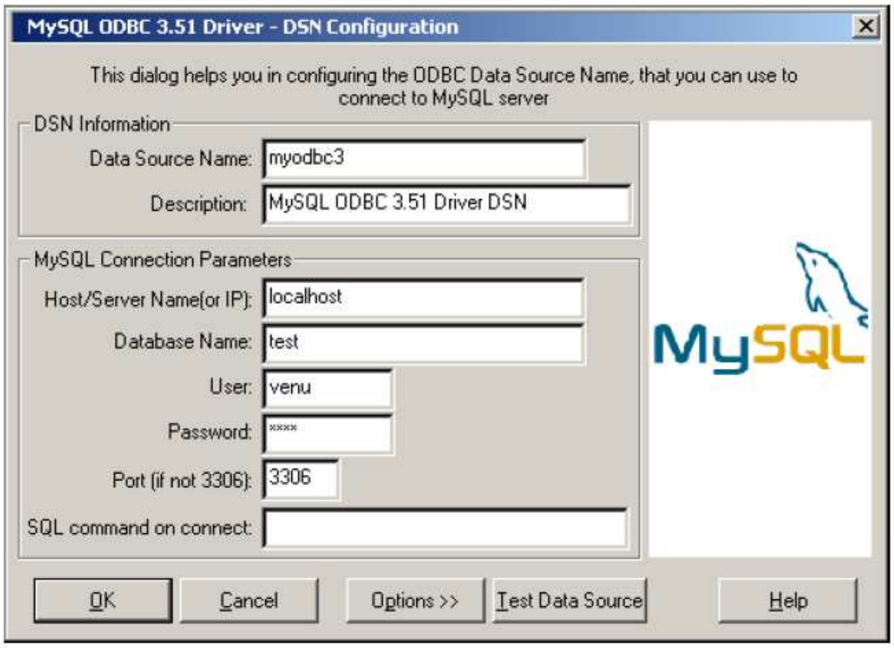


User Data Sources:

Name	Driver
ibm	IBM DB2 ODBC DRIVER
myodbc	MySQL Driver
myodbc3	MySQL ODBC 3.51 Driver
myodbc3-test	MySQL ODBC 3.51 Driver
test	MySQL Driver

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

Figure 40. ODBC Configuration for MySQL[11]


MySQL ODBC 3.51 Driver - DSN Configuration

This dialog helps you in configuring the ODBC Data Source Name, that you can use to connect to MySQL server

DSN Information

Data Source Name: myodbc3

Description: MySQL ODBC 3.51 Driver DSN

MySQL Connection Parameters

Host/Server Name(or IP): localhost

Database Name: test

User: venu

Password: ****

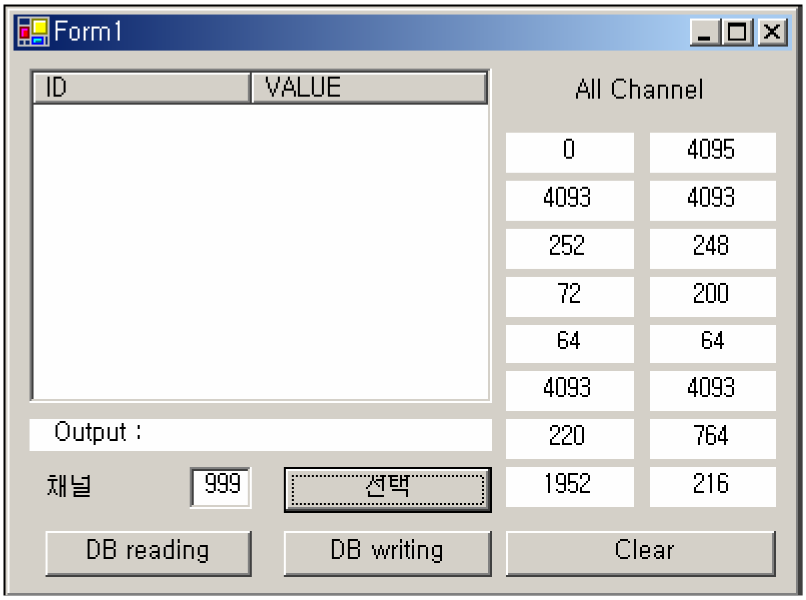
Port (if not 3306): 3306

SQL command on connect:

3.3.3. User Interface Implementation

C# version of user interface program interlocking with database and service program has been coded for system verification. Real-time channel value, weight variations of strain gauge sensor, can be retrieved over Internet connection. Figure 41 shows the output screen of program when all channel selection value 999 had been entered.

Figure 41. All Channel Output



0	4095
4093	4093
252	248
72	200
64	64
4093	4093
220	764
1952	216

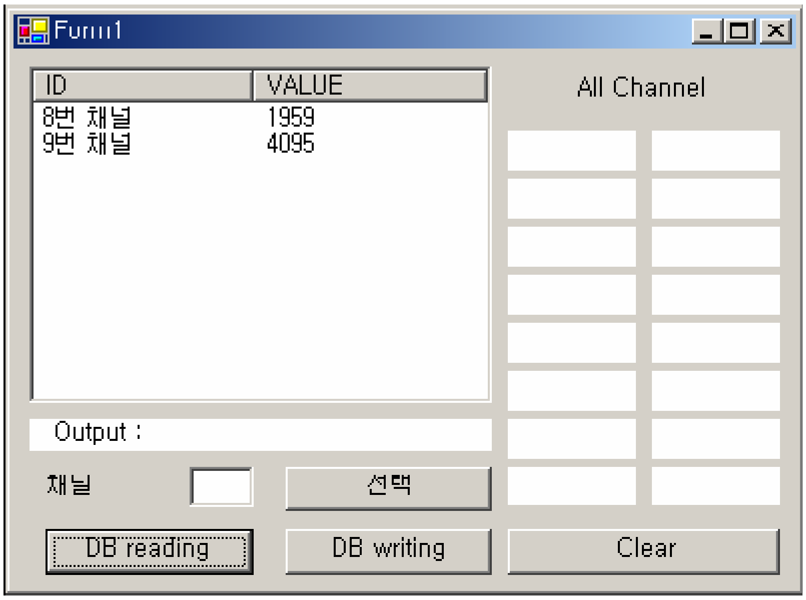
Following code is a core part of this process. There are two connections for 12 bits data retrieval since RS232 communication is based on 1 byte. As I mentioned in process module part, asynchronous problem of 2 bytes serial communication has been prevented gracefully.

```
data = Encoding.Default.GetBytes("A");
server.SendTo(data, data.Length, SocketFlags.None, ipep);
recv_size = server.ReceiveFrom(data, ref remote);
int rst = data[0];
data = Encoding.Default.GetBytes("Z");
server.SendTo(data, data.Length, SocketFlags.None, ipep);
recv_size = server.ReceiveFrom(data, ref remote);
rst = rst + data[0] * 256;
```

```
label2.Text="";
label2.Text=Convert.ToString(rst)
```

An example of ODBC database connection has been presented on Figure 42 with associated source code.

Figure 42. Saving Retrieved Data into Database



```
private void button1_Click(object sender, System.EventArgs e)
{
  string source = "DRIVER={MySQL ODBC 3.51 Driver};" +
                  "SERVER=220.67.220.150;" +
                  "DATABASE=test;" +
                  "UID=root;" +
                  "PASSWORD=;" +
                  "OPTION=3";

  OdbcConnection MyConnection = new OdbcConnection(source);
  MyConnection.Open();

  OdbcDataAdapter adapter = new OdbcDataAdapter();
  DataSet ds = new DataSet();

  string sql1 = "insert into embedded(name, input)
                 values('"+textBox1.Text+"번채널"+"','"+label2.Text+"')";
  adapter.SelectCommand = new OdbcCommand( sql1, MyConnection );

  adapter.Fill( ds );

  MyConnection.Close();
}

private void button2_Click(object sender, System.EventArgs e)
{
  listView1.Items.Clear();

  string source = "DRIVER={MySQL ODBC 3.51 Driver};" +
                  "SERVER=220.67.220.150;" +
                  "DATABASE=test;" +
                  "UID=root;" +
                  "PASSWORD=;" +
                  "OPTION=3";

  OdbcConnection MyConnection = new OdbcConnection(source);
  MyConnection.Open();

  OdbcDataAdapter adapter = new OdbcDataAdapter();
  DataSet ds = new DataSet();

  string sql = "SELECT * FROM embedded";
  adapter.SelectCommand = new OdbcCommand( sql, MyConnection );
  adapter.Fill( ds );

  DataTable dt = ds.Tables[0];

  foreach (DataRow row in dt.Rows)
  {
    ListViewItem item = new ListViewItem( row[0].ToString() );
    item.SubItems.Add( row[1].ToString() );

    listView1.Items.Add( item );
  }

  MyConnection.Close();
}
```

Chapter 4 Conclusion

This paper describes one type of wireless sensor network in which sensor embedded system, RF communication system, and remote measuring system over Internet are mixed and balanced. In order to overcome the time sequence problem among different device parts, a total communication system using handshaking method has been developed. This kind of system can be used in many new applications with minor changes, ranging from environment monitoring to industrial sensing. In fact, the applications are only limited by our imagination.

Figure 43. 1 Channel Sensor Module

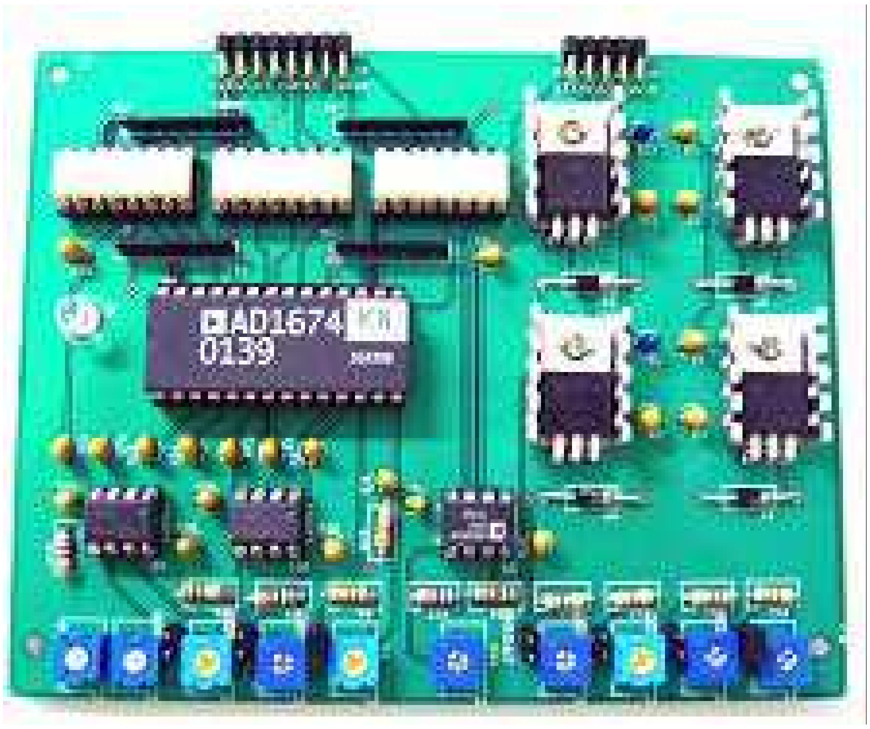


Figure 44. Combined Controller Module with Bluetooth Tranceiver

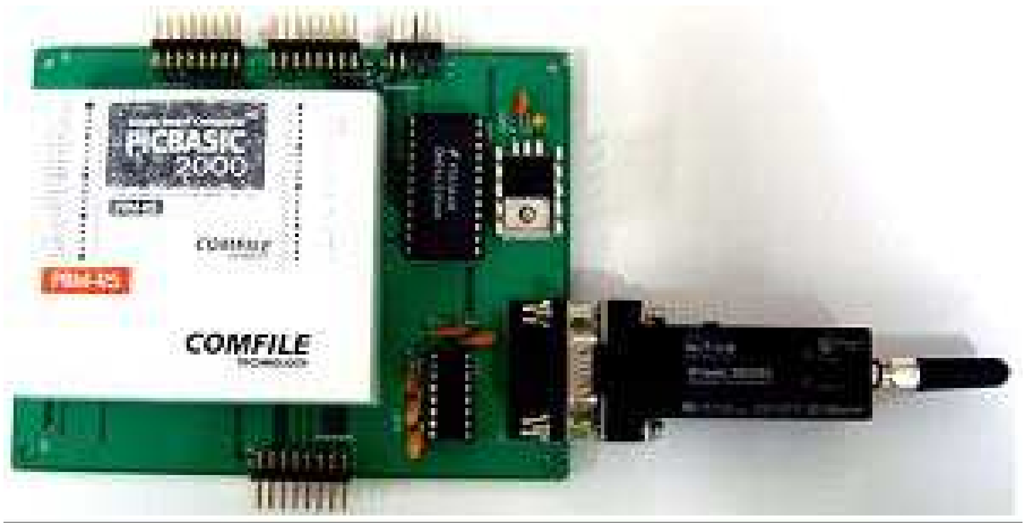
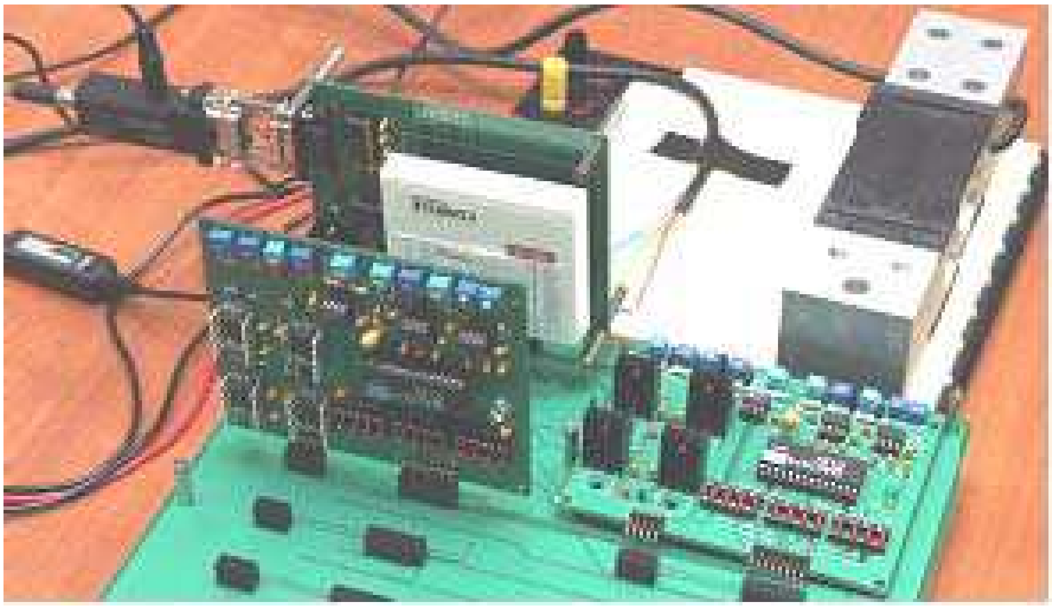


Figure 45. Assembled System in Testing Lab



References

[1] S. Arimoto, “Linear controllable system,” Nature, vol. 135, pp. 18-27, July, 1990.

[2] “21 ideas for the 21st century,” Business Week, pp. 78-167, Aug. 30, 1999.

[3] Proceedings of the Distributed Sensor Nets Workshop. Pittsburgh, PA: Dept. Comput. Sci., Carnegie Mellon Univ., 1978.

[4] “Distributed sensor networks,” MIT Lincoln Laboratory, Lexington, MA, Rep. No. ESD-TR-88-175, 1986.

[5] LC4103 serious Specification, http://www.andk.co.kr/product/rod/no_lc4103.php

[6] AD620 Instrumentation Amp Data Sheets http://www.analog.com/UploadedFiles/Data_Sheets/897653854AD620_g.pdf

[7] AD1674 12-Bit, 100kSPS, Complete ADC Data Sheets http://www.analog.com/UploadedFiles/Data_Sheets/346669145AD1674_c.pdf

[8] Tibbo Technology DS100 Serial Device Server http://www.tibbo.com/ds100.php

[9] Promi-DS wireless serial communicationn manual http://www.initium.co.kr/bizdata/userguide/Promi-SD_User%20manual_V2.5.pdf

[10] Comfile Technology PICBASIC Databook http://www.comfile.co.kr/download/pb/PBMAN10.pdf

[11] Microsoft Developer Network http://msdn.microsoft.com/

[12] Steven W.Richard, “TCP/IP Illustrated Volume 1: The Protocols,” Addison-Wesley, 1993

[13] 한국AND LC4103 Series http://www.andk.co.kr/product/rod/no_lc4103.php