GPRS library for

G-4500 embedded controller

User Manual

Warranty

All products manufactured by ICP DAS are warranted against defective materials for a period of one year from the date of delivery to the original purchaser.

Warning

ICP DAS assumes no liability for damages consequent to the use of this product. ICP DAS reserves the right to change this manual at any time without notice. The information furnished by ICP DAS is believed to be accurate and reliable. However, no responsibility is assumed by ICP DAS for its use, or for any infringements of patents or other rights of third parties resulting from its use.

Copyright

Trademark

The names used for identification only may be registered trademarks of their respective companies.

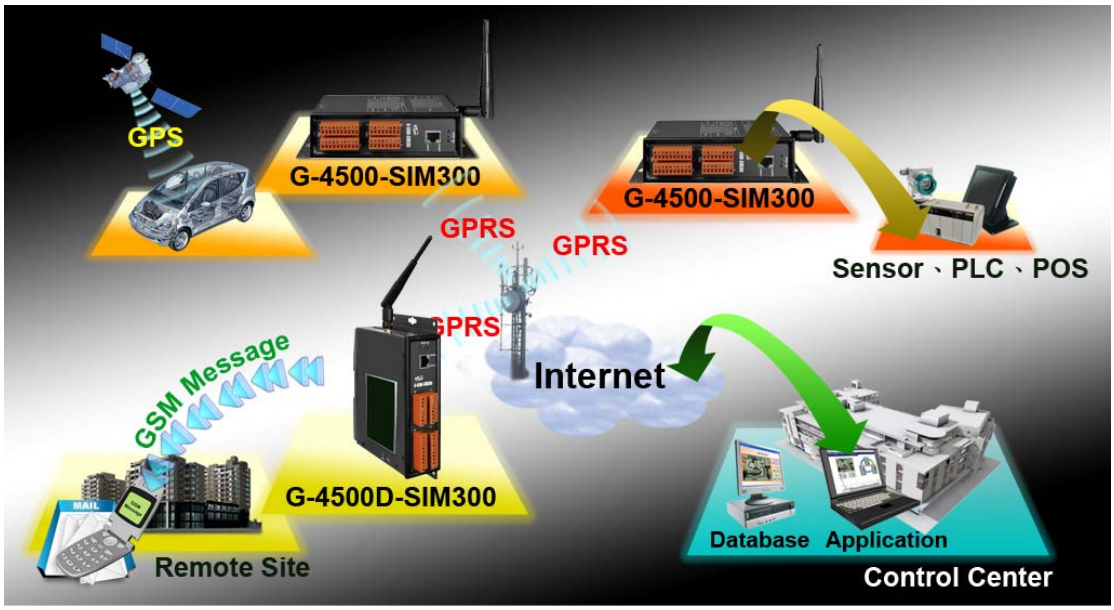
Table of Contents

Chapter 1 Introduction

1.1 GPRS and G-4500 series embedded controller

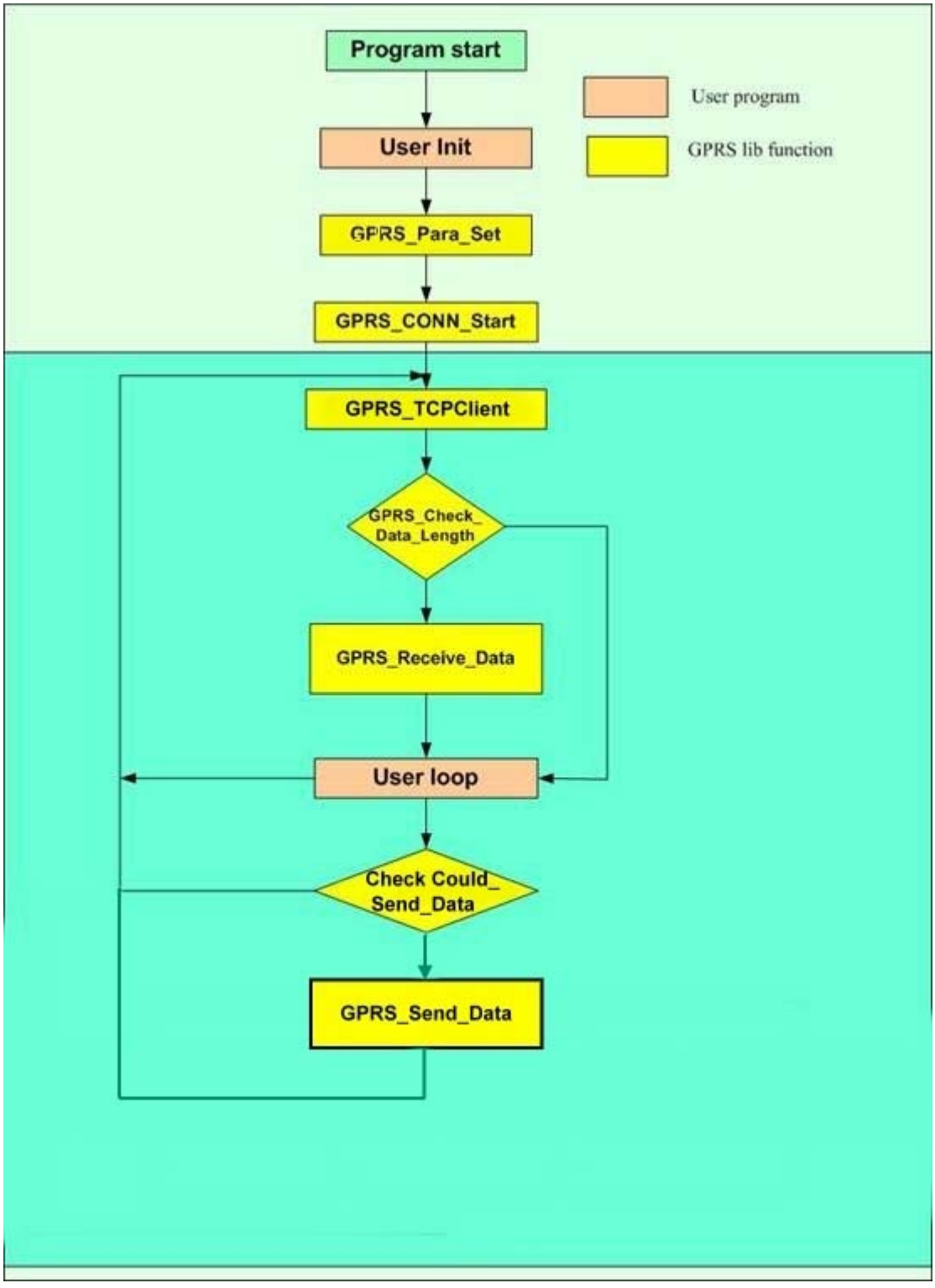
The General Packet Radio Service (GPRS) is a new nonvoice value added service that allows information to be sent and received across a mobile telephone network. It supplements today's Circuit Switched Data and Short Message Service. GPRS is NOT related to GPS (the Global Positioning System), a similar acronym that is often used in mobile contexts. GPRS facilitates instant connections whereby information can be sent or received immediately as the need arises, subject to radio coverage. No dial-up modem connection is necessary. This is why GPRS users are sometimes referred to be as being "always connected". Immediacy is one of the advantages of GPRS (and SMS) when compared to Circuit Switched Data. High immediacy is a very important feature for time critical applications.

ICP DAS provides the GPRS library for G-4500 embedded controller. The library is an easy way to applying the GPRS service in the embedded controller. Otherwise, ICP DAS supports many IO modules and GPS modules for users. Therefore, there are many application architectures to apply in the system. Or users can integrate other controller system with GPRS library. The follows is a standard application architecture.



1.2 Design flow chart

It is very difficult to develop an embedded controller program with GPRS communications. However, users are allowed to make those developments quickly and easily by using the ICP DAS's GPRS lib. The design flow is as follows.



Chapter 2 GPRS Library

2.1 Function and variable definition

➢ GPRS Function:

Function definition	Describe
Extern int Could_Send_Data;	The global variable for could send data.
extern char GPRS_Local_IP[20];	Get Local IP address.
int GPRS_Check_MODEM(void);	Check the modem whether is normal.
int GPRS_Para_Set(struct str_GPRS_setting GPRS_setting);	Setting the GPRS communication information.
void GPRS_CONN_Start(void);	Initial connect the remote server argument.
int GPRS_Chack_Status(void);	Check GPRS connection status.
int GPRS_TCPClient(void);	Start to connect the remote server. The function must be called anytime.
int GPRS_Send_Data(char *sData);	Send the data to the remote server.
int GPRS_Check_Data_Length(void);	Check the length of the received data in the buffer
int GPRS_Receive_Data(char *data);	Receive the data from the remote server.
int GPRS_PPP_Close(void);	Terminate the GPRS connection.
int GPRS_TCP_Close(void);	Stop TCP/IP connection to the remote server.
int GPRS_PPP_RECONN(void);	Reconnect the PPP connection and reconnect TCP/IP.
int GPRS_TCP_RECONN(void);	Reconnect the TCP/IP.
int GPRS_Modem_Reset(void);	Reset the modem.
int GPRS_Check_CSQ(void);	Check GSM system CSQ.
int RTC_GPRS_Time(unsigned char *GMT)	This function allows users to set time of MiniOS7 to GPRS modem

- SMS Function:

Function definition	Describe
int SMS_Select_Mode(int mode);	Select the SMS mode.
int SMS_TextMode_Send (SMS_Setting str_SMS);	Send a message to assign phone number by TextMode.
int SMS_PDUMode_Send (SMS_Setting str_SMS, int data_length);	Send a message to assign phone number by PDU mode.
int SMS_Del_Msg(int index);	Delete assign index message.
int SMS_Message_Read(int index,char *szResult);	Read assign index message.

2.2 Struct define

str_GPRS_setting:

Variable definition	Describe
char *phone;	The phone number of ISP
char *user_name;	User name for connecting to ISP
char *password;	Password for connecting to ISP
char *GPRS_server1	The remote server's IP
char *GPRS_server2	The remote server's IP
char modem_com;	The com port for GPRS modem
int RemotePort;	Send the data to the remote server
int debug_port;	The com port for debug
unsigned long modem_baud;	Set the modem baud rate
unsigned long debug_baud;	Set the debug baud rate
int Enable_debug;	Print out debug message or not.
char ExtraStr[50];	ExtraStr
int AutoRedial	When ISP hang off this connection, if re-connect or no.

SMS_Setting:

Variable definition	Describe
char phone[20];	Send a message to this phone number.
char content[200];	Send a content of message.

2.3 Global Variable

- **Extern int Could_Send_Data**

Description:

The global variable Could_Send_Data is shown the command status. If the value is not equal 1, users can not call the following function: GPRS_TCP_Close, GPRS_Send_Data, GPRS_Receive_Data and GPRS_PPP_Close.

0	Disconnect or in send data
1	Could send data

- **Extern char GPRS_Local_IP[20];**

Description:

Get GPRS Loacl IP address.

2.4 Function

2.4.1 GPRS_Check_MODEM

Prototype: int GPRS_Check_MODEM(void)

Description:

If modem is not in GPRS connection, users can call the function to check the modem whether is normal.

Parameter: None

Return:

0 -> OK

Others -> The modem does not response. Check the modem and the baud rate.

2.4.2 GPRS_Para_Set

Prototype: int GPRS_Para_Set(struct str_GPRS_setting GPRS_setting)

Description:

This function allows users to setting the parameters needed in GPRS.

Parameter:

```
struct str_GPRS_setting
{
    char *phone;                // ex. "4121234"
    char *user_name;            // ex. "ufmika"
    char *password;             // ex. "abc123"
    char *GPRS_server1;         // ex. "192.168.0.254"
    char *GPRS_server2;         // ex. "192.168.0.254"
    char modem_com ;
    int   RemotePort ;
    unsigned long modem_baud;
    unsigned long debug_baud;
    int   debug_port;           //debug message com port
    char ExtraStr[50];          // ex.INTERNET : Taiwan     CMNET : China
    int   Enable_debug;         //show the debug message
    int   AutoRedial;           //0：no re-connect     1：re-connect
};
```

Return:

0 -> OK

Others -> The configuration setting have error

Example:

```
struct str_GPRS_setting rGPRS;
  rGPRS.phone = "*99***1#"       // *** use your own ***
  rGPRS.user_name = "";          // *** use your own ***
  rGPRS.password = "";           // *** use your own ***
  rGPRS.GPRS_server1 = "218.164.78.151";     // *** choose server ***
  rGPRS.GPRS_server2 = "218.164.78.151";     // *** choose server ***
  rGPRS.modem_com=1;
  rGPRS.RemotePort=10000;
  rGPRS.debug_baud=115200L;
  rGPRS.modem_baud=115200L;
```

```
rGPRS.debug_port=1;
rGPRS.AutoRedial=1;
rGPRS.Enable_debug=1;
rGPRS.ExtraStr="INTERNET"
ret=GPRS_Para_Set(rGPRS);
```

2.4.3 GPRS_CONN_Start

Prototype: void GPRS_CONN_Start(void)

Description:

This function allows users to initial dial connection to ISP argument. Before using the function, the GPRS_Para_Set() must be called.

Parameter: None

Return: None

Example

```
…
  ret=GPRS_Para_Set(rGPRS);
  GPRS_CONN_Start();
…
```

2.4.4 GPRS_Check_Status

Prototype:int GPRS_Check_Status(void)

Description:

This function allows users to check connection state.

Parameter: None

Return:

0 IP INITIAL
1 IP START
2 IP CONFIG
3 IP IND
4 IP GPRSACT
5 IP STATUS
6 TCP CONNECTING
7 IP CLOSE
8 CONNECT OK
9 PDP DEACT

2.4.5 GPRS_TCP_Client

Prototype: int GPRS_TCPClient(void)

Description:

This function allows users to connect remote TCP server. The function must be called in a while loop.

Parameter: None

Return:

1-79— GPRS dialing
80 — Connect OK
-1 — TIMEOUT
-2 —System no ready
-3 —Send Data TimeOut
-4 —Send Data ERROR

Example

```
while(1)
  {
    …
    ret= GPRS_TCPClient (); // must be in loop
     …
    }
```

2.4.6 GPRS_Send_Data

Prototype: int GPRS_Send_Data(char *sData)

Description:

This function allows users to send data to the server in the Internet. Users can check the Could_Send_Data. When Could_Send_Data is equal 1, the data can be sent to the server via GPRS. If the variable is not equal 1, the function could not be called. Send data isn't more than 1024 bytes.

Parameter:

char *sData -> The sent data point address.

Return:

0 -> The data can be send to the server.

-1-> The data more than 1024 bytes.(ERROR)

Note: The terminal character of sData is 0x0.

Example:

```
if (Could_Send_Data ==1 ){
    sprintf(tmpp,"Send Data…\n\r");
    ToComStr(rGPRS.debug_port,tmpp);
    GPRS_Send_Data("222");
    }
```

2.4.7 GPRS_Check_Data_Length

Prototype: int GPRS_Check_Data_Length(void)

Description:

This function allows users to check the length of data from the server in the Internet. Users can check the number of data by calling the GPRS_Check_Data_Length function. If the number of data is not zero, users can use GPRS_Receive_Data to get the data.

Parameter: None

Return:

Receive data length from the remote server.

Example:

```
if (GPRS_Check_Data_Length()>0){
    GPRS_Receive_Data(data);
    sprintf(tmpp,”receive=%sn\r”,data);
    ToComStr(rGPRS.debug_port,tmpp);
}
```

2.4.8 GPRS_Receive_Data

Prototype: int GPRS_Receive_Data(char *data)

Description:

This function allows users to receive data from the server in the Internet. Users can check the number of data by calling the GPRS_Check_Data_Length function. If the number of data is not zero, users can use GPRS_Receive_Data to get the data.

Parameter:

char *data -> The data from the remote server.

Return:

0 -> The data can be received succes

-1-> No receive any data

Example:

```
if (GPRS_Check_Data_Length()>0){
    GPRS_Receive_Data(data);
    sprintf(tmpp,”receive=%sn\r”,data);
    ToComStr(rGPRS.debug_port,tmpp);
}
```

2.4.9 GPRS_ PPP_Close

Prototype: int GPRS_PPP_Close(void)

Description:

This function allows users to tell ISP to terminate the PPP connection. If users want to connect the PPP connection after terminate the PPP connection, we must to call GPRS_PPP_RECONN.

Parameter: None

Return:

0 -> Success

Others -> The modem does not response.

Example:

```
…
  GPRS_PPP_Close ();
  DelayMs(500);
  RestoreCom(4); //Terminate GPRS connection
  …
```

2.4.10 GPRS_ TCP_Close

Prototype: int GPRS_TCP_Close(void)

Description:

This function allows users to terminate the TCP/IP connection. If users want to connect the TCP/IP connection after terminate the connection by calling GPRS_TCP_Close, must to call GPRS_TCP_RECONN.

Parameter: None

Return:

0 -> Success

Others -> The modem does not response.

Example:

```
GPRS_PPP_Close ();
DelayMs(500);
```

2.4.11 GPRS_PPP_RECONN

Prototype: int GPRS_PPP_RECONN(void)

Description:

This function allows users to reconnect the PPP connection.

Parameter: None

Return:

0 -> Success

Others -> The modem does not response.

2.4.12 GPRS_TCP_RECONN

Prototype: int GPRS_TCP_RECONN(void)

Description:

This function allows users to reconnect the TCP/IP connection.

Parameter: None

Return:

0 -> Success

-1->No establish PPP connection

Others -> The modem does not response.

2.4.13 GPRS_Modem_Reset

Prototype: GPRS_Modem_Reset(void)

Description:

This function allows users to reset the modem.

Parameter: None

Return:

0 -> The modem reset success.

Others -> The modem reset failed and must be re-call the function.

2.4.14 GPRS_Check_CSQ

Prototype: int GPRS_Check_CSQ(void)

Description:

This function allows users to check GSM CSQ(Signal Quality Report).

Parameter: None

Return: CSQ value

2.4.15 RTC_GPRS_Time

Prototype: int RTC_GPRS_Time(unsigned char *GMT)

Description:

This function allows users to set time of MiniOS7 to GPRS modem.

Parameter:

unsigned char *GMT -> Set the city GMT.

Format : ±zz Ex.Taiwan GMT= +08

Return:

0 -> The modem reset success.

-1 -> The modem reset failed and must be re-call the function.

2.4.16 SMS_Select_Mode

Prototype: int SMS_Select_Mode(int mode)

Description:

Select the SMS mode.

Parameter:

Mode : 0 PDU mode(UCS2)

1 Text mode

Return:

0 -> Setting success.

-5 -> TimeOut, Error.

2.4.17 SMS_TextMode_Send

Prototype: int SMS_TextMode_Send(SMS_Setting str_SMS)

Description:

Send a message to assign phone number by TextMode.

Parameter:

SMS_Setting str_SMS(please see Struct define chapter 2.2)

Return:

0 -> Send success.

-1 -> Content of message too long (Max. 160 character), Error.

-5 -> TimeOut, Error.

Note:

Must be change mode to Text Mode before called SMS_TextMode_Send function and content of message Max. **160** character.

Example:

```
SMS_Setting SMS_Text;
sprintf(SMS_Text.phone,”09xx123456”);
sprintf(SMS_Text.content,”hello word!!”);

iRet=SMS_Select_Mode(1); //Select SMS Text mode
Print("iRet=> %d\r\n", iRet);
iRet=SMS_TextMode_Send(SMS_Text);    //Send a message
```

2.4.18 SMS_PDUMode_Send

Prototype: int SMS_PDUMode_Send(SMS_Setting str_SMS, int data_length)

Description:

Send a message to assign phone number by PDU mode.

Parameter:

SMS_Setting str_SMS(please see Struct define chapter 2.2)

Data_lentgh : content of message length.

Return:

0 -> Send success.

-1 -> Content of message too long (Max. 70 UCS-2), Error.

-5 -> TimeOut, Error.

Note:

Must be change mode to PDU Mode before called SMS_PDUMode_Send function and content of message Max. **70** character.

Example:

```
SMS_Setting SMS_PDU;
sprintf(SMS_PDU.phone,”09xx123456”);
sprintf(SMS_PDU.content,"%c%c%c%c",0x4F,0x60,0x59,0x7D);
 //   0x4F,0x60       0x59,0x7D   你好(Unicode)

iRet=SMS_Select_Mode(0); //Select SMS Text mode
Print("iRet=> %d\r\n", iRet);

iRet=SMS_PDUMode_Send(SMS_PDU,4);   //Send a message
```

2.4.19 SMS_Del_Msg

Prototype: int SMS_Del_Msg(int index)

Description:

Delete assign index message.

Parameter:

Index: SIM card index.

Return:

0 -> Delete success.

-5 -> TimeOut, Error.

2.4.20 SMS_Message_Read

Prototype: int SMS_Message_Read(int index,char *szResult)

Description:

Read assign index message.

Parameter:

Index: SIM card index.

szResult: Return read content of message.

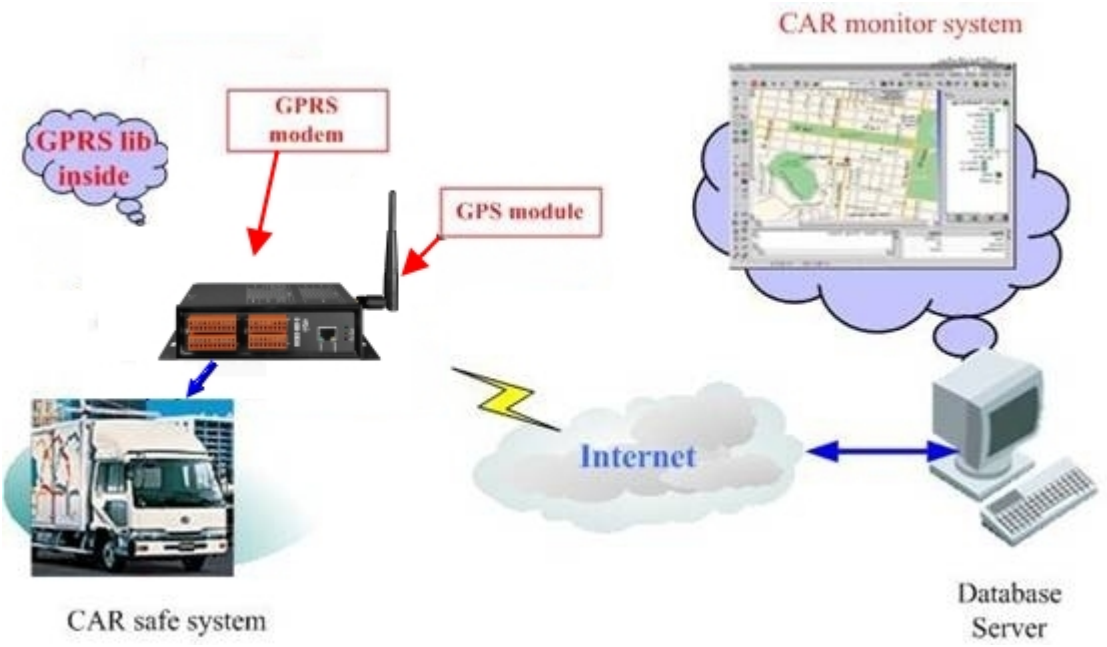
Return:

0 -> Read success.

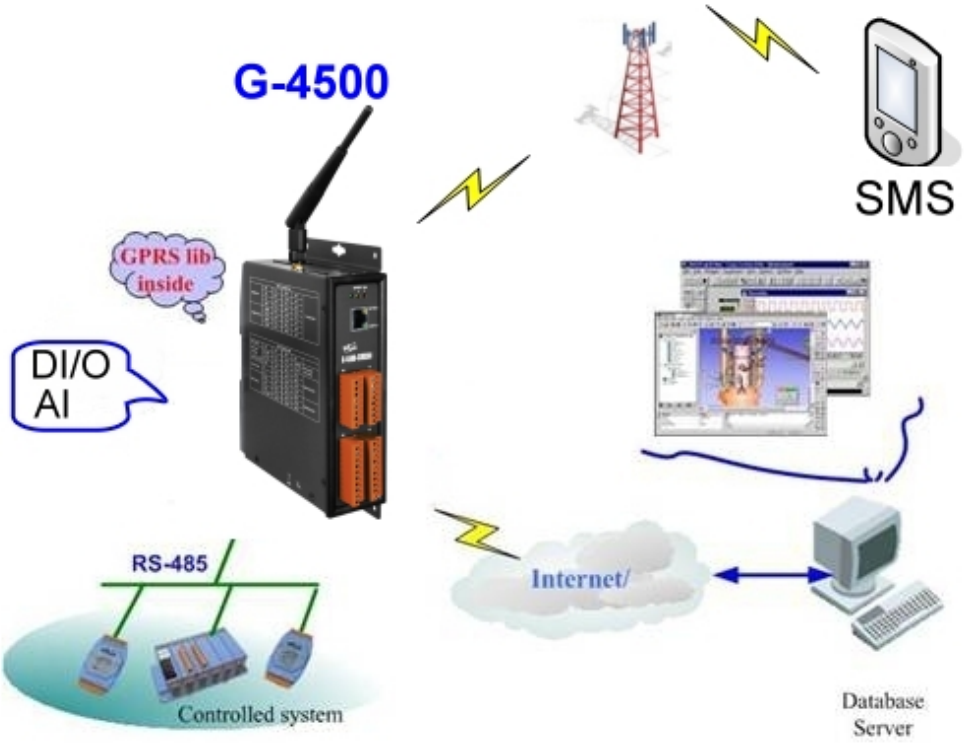
-5 -> TimeOut, Error.

Chapter 3 Application architecture

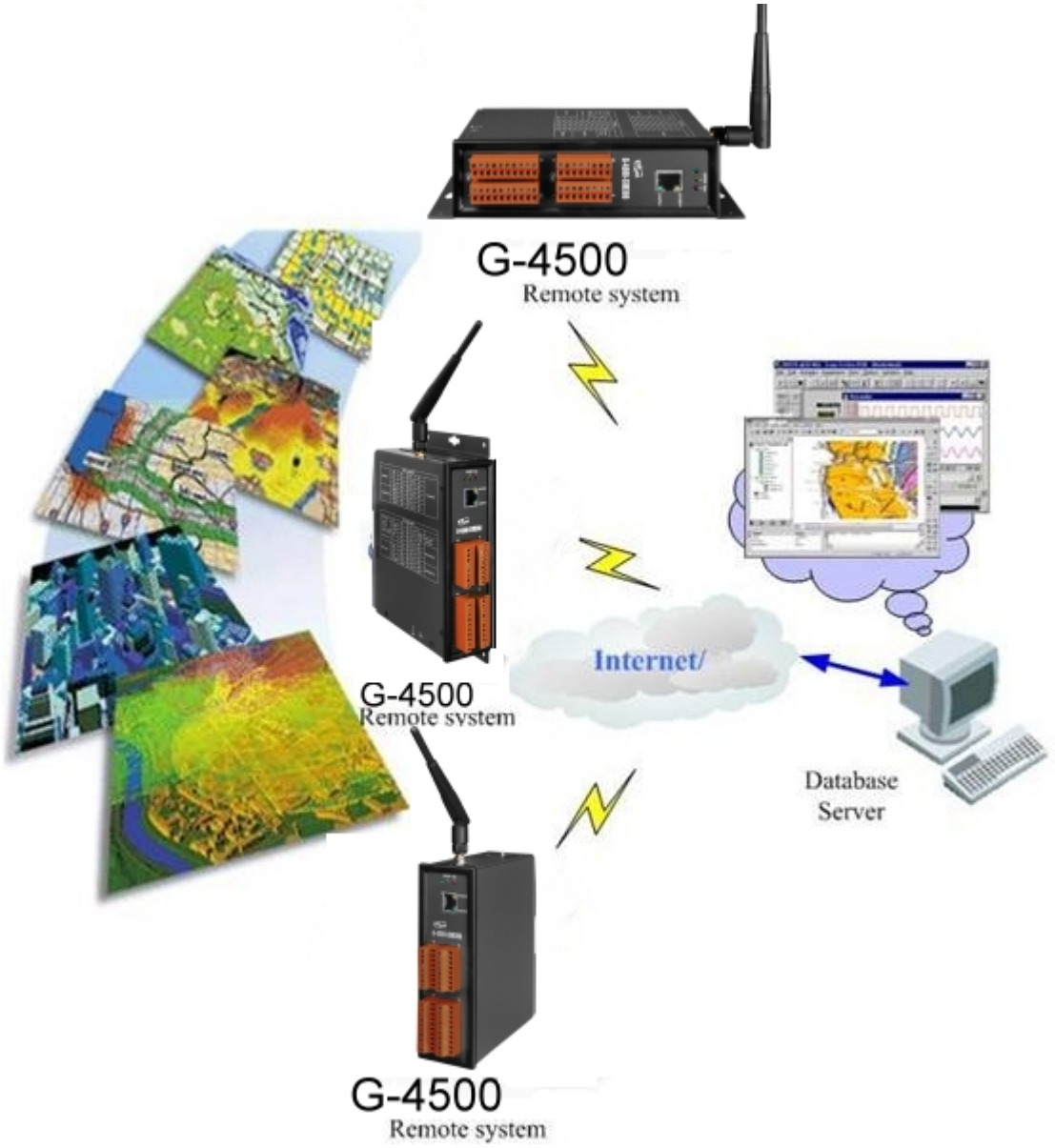
3.1 Car Monitor System



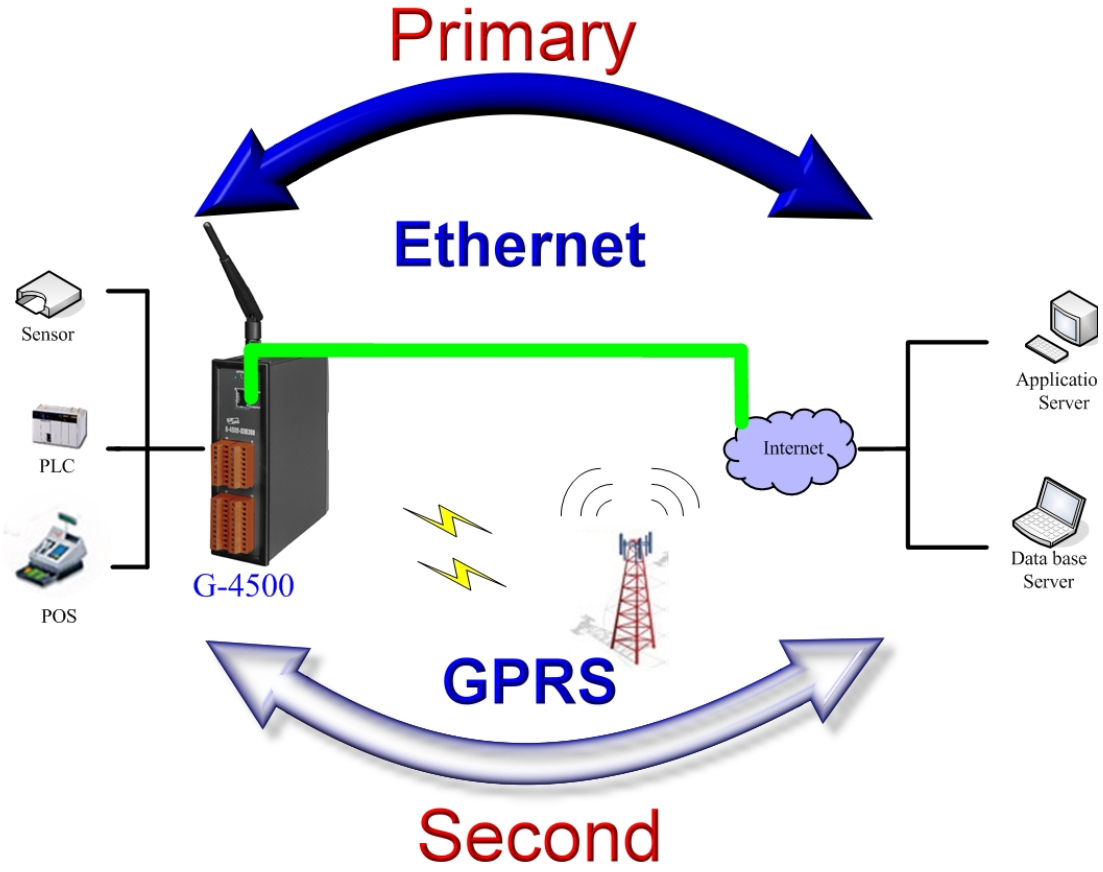
3.2 Remote Control System



3.3 GIS system



3.4 Redundance Communication System



Version Record

Version	By	Date	Description
1.0.1	Anold	2004/11/15	
1.0.2	Yide	2008/06/04	
1.0.3	Yide	2008/06/17	
1.0.4	Yide	2008/07/07	
1.0.5	Yide	2008/10/30	
1.06	Yide	2009/03/09	
1.07	Yide	2010/03/25	