Third Prize

Nios II-Based Multiple-Core Intelligent Traffic On-Vehicle Terminal

Institution: **Xi'an Institute of Post and Telecommunications**

Participants: **Xiao-Wu Chen, Li-Bin Liu, and Fang-Fang Zhang**

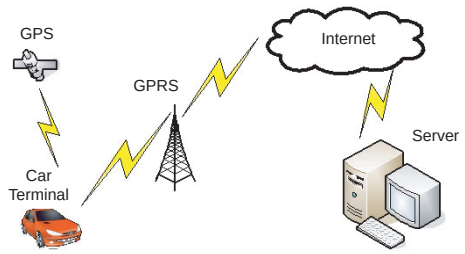
Instructor: **Jun Liu**

Design Introduction

As China's automobile industry has developed in recent years, more people are traveling in private cars to explore tourist sites. The traditional method of using maps to get to a destination does not provide enough objective details. Instead, car navigation systems are the future.

The navigation systems currently in the market provide information about roads and locations, but do not tell the driver about traffic jams, available parking lots, etc. Using a PC-based, real-time, Internet road inquiry system is not convenient or practical. To solve this problem, we designed an embedded system, the Nios II®-based multiple-core intelligent vehicle terminal. The system is convenient and practical, and can obtain real-time information, such as traffic jam and parking lot information, by wirelessly connecting to a background server via the Internet. Figure 1 shows the system schematic.

Figure 1. System Schematic Diagram



We chose the Nios II processor for our design for the following reasons:

- Using multiple Nios II processors solves a variety of problems, such as low microprocessor (MCU) processing speed, limited peripheral resources, complicated interface configuration, complex hardware, and software design and programming. With two Nios II processors, we can allocate the system controls and peripheral access, which greatly decreases the software workload.

- Programmable logic devices (PLDs) are outstanding in lowering project development costs and for trial chip production. As we developed and built the design, we could program and reprogram the PLD.

Our Nios II-based multiple-core intelligent vehicle terminal is suitable for a variety of vehicles, but we mainly target mid-level and high-end cars.

Function Description

The Nios II-based multiple-core intelligent vehicle terminal implements the following functions:

- *Global positioning system (GPS)*—The system uses a GPS to note the vehicle's position accurately.

- *Navigation*—Users can scale and translate the map, find a location, and select the destination. The system displays the vehicle's real-time surroundings (roads and buildings).

- *Real-time road message updates*—As the user is driving, the system displays real-time traffic jam information on the e-map, reminding the driver to avoid it.

- *Touch screen operation*—With the touch screen, the driver can interact with the system at any time to complete any operation.

- *Client-server mode*—The driver can obtain locale information from the background server, such as parking lots, hotels, hospitals, shopping malls, schools, etc.

Performance Parameters

The following list describes the performance and specifications for our system.

- Power supply
  - DC voltage: 12 V
  - Operating current: 1 A
  - Power consumption: 12 W
- Operating temperature: 0 to 50° C
- Hitachi IS61LV6416-10T LCD
  - LCD resolution: 640 x 480
  - LCD display space: 19 cm (7.5 inches)

  - LCD display color depth: 8-bit RGB
  - Viewing angle: 80 degrees
- GS1100
  - Operating voltage: 5 V
  - Frequency: 1 Hz
  - Data transmission velocity: 9,600 bits per second (bps)
  - Positioning error: 15 m
  - Features a built-in passive antenna, low power consumption, and sensitivity up to -153 dB/mW (dBm)
- Siemens MC35I—Double-frequency general packet radio service (GPRS) module, circuit switched data (CSD) maximizing to 14.4 kilobits per second (Kbps), unstructured supplementary service data (USSD), and nontransparent mode.
- Hitachi IS61LV6416-10T touch screen
  - 19 cm (7.5 inches), same as the LCD
  - 4-wire resistive touch screen
- Epson S1D 13503F LCD controller:
  - Available modes: 320 x 240 x 256 colors, 640 x 480 x 16, 8, or 4 colors, 1,024 x 768 x 2 colors
  - Available memory size: 128 Kbytes maximum
- MXB7843 touch screen controller—Programmable 8-12 route analog-to-digital (A/D) conversion, 2-route input serial peripheral interface (SPI)

Design Architecture

The system design is divided into hardware and software.

The hardware design includes the following items:

- Nios II dual-core configuration
- Peripheral driver circuit board design
- Self-defined expanded bus
- GPS module
- GPRS module
- LCD module, touch screen module, etc.

The software design includes the following items:

- Self-defined custom instruction
- Embedded real-time operating system µC/OS-II
- Embedded graphics interface development software µC/GUI
- Self-defined map messages system
- GPS data receiving and serial driver
- GPRS data receiving and delivering, network communication protocols (e.g., PPP, TCP/IP)

Figure 2 shows the software structure.

Figure 2. Software Structure

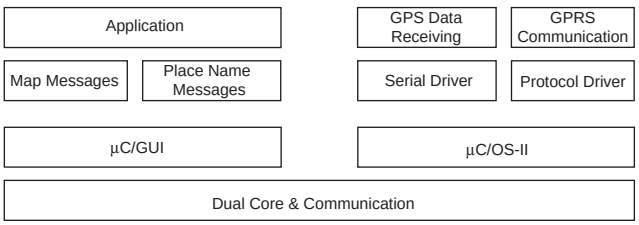


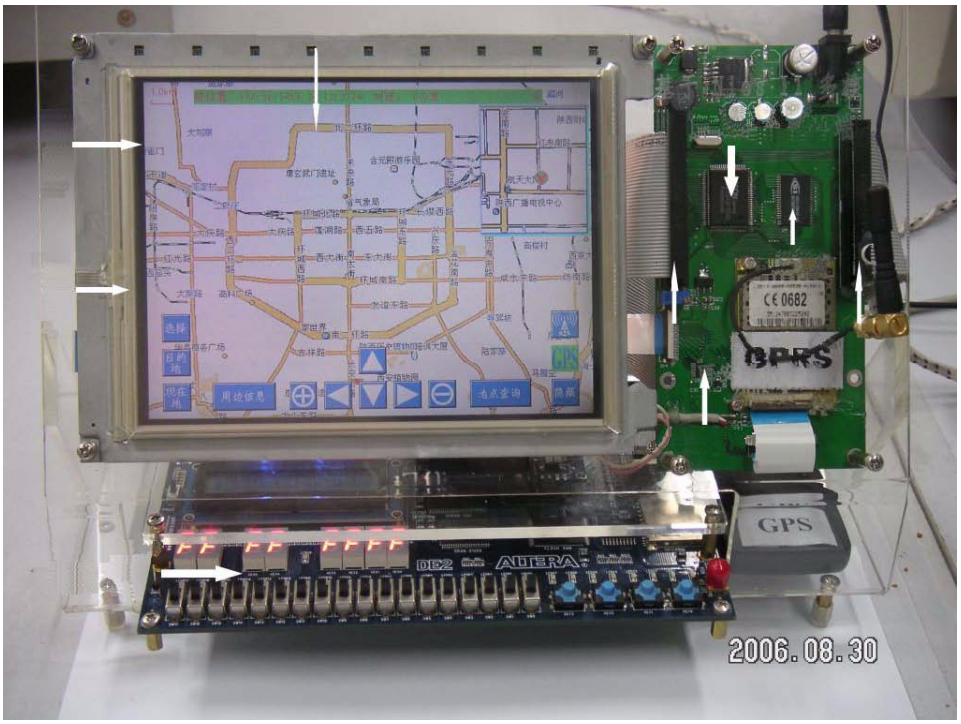
Table 1 shows the functions and main technologies used in our Nios II-based multiple-core intelligent vehicle terminal system.

Table 1. Functions and Technologies

Function	Implementation
Navigation Location enquiry function Function of real-time update of the road messages	LCD Touch screen Self-defined expanded bus Self-defined map messages µC/GUI, µC/OS-II
Dynamic positioning function	Serial communication
Client-server mode	GPRS wireless communication Self-defined custom instruction Network protocol Database
Dual core	Nios II CPU 1 operates the application (GUI) Nios II CPU 2 operates the GPS and GPRS modules

Figure 3 shows the system hardware.

Figure 3. System Hardware



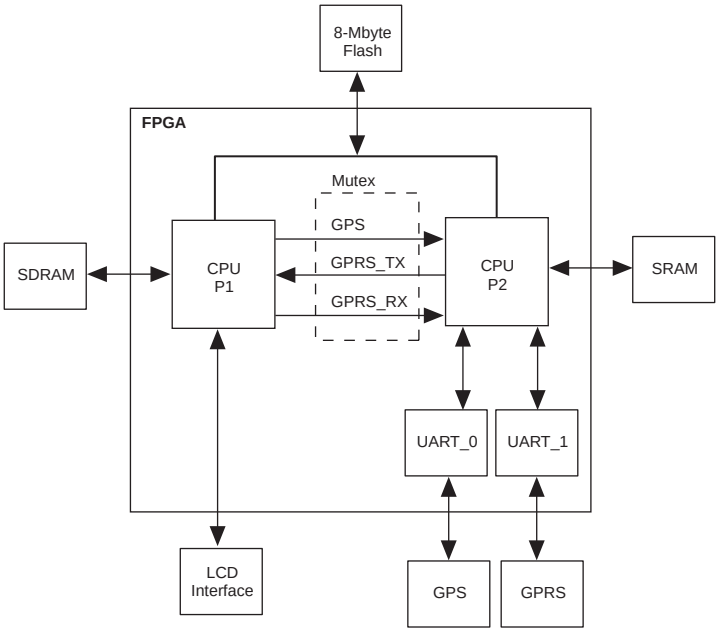
System Structure

The system provides customers with real-time traffic jam information. In client-server mode, the system stores static map information in the vehicle terminal and delivers dynamic information using the background server and the wireless communication module.

Hardware Structure

Figure 4 shows the hardware design. CPU 1 operates the application (GUI), CPU 2 operates the GPS and GPRS modules. Three resistive cores use CPU 1 and CPU 2 to communicate. The `GPS` signal provides communication between the GPS module and the key GUI. The GPRS uses the `GPRS_TX` signal to deliver data to the GUI. The GUI uses the `GPRS_RX` signal to deliver data to the GPRS.

Figure 4. Hardware System Structure



Software Module Structure

Figure 5 shows the software module interface digram. The interface lets the application (GUI), GPS module, and GPRS module communicate.

Figure 5. Software Module Interface Diagram

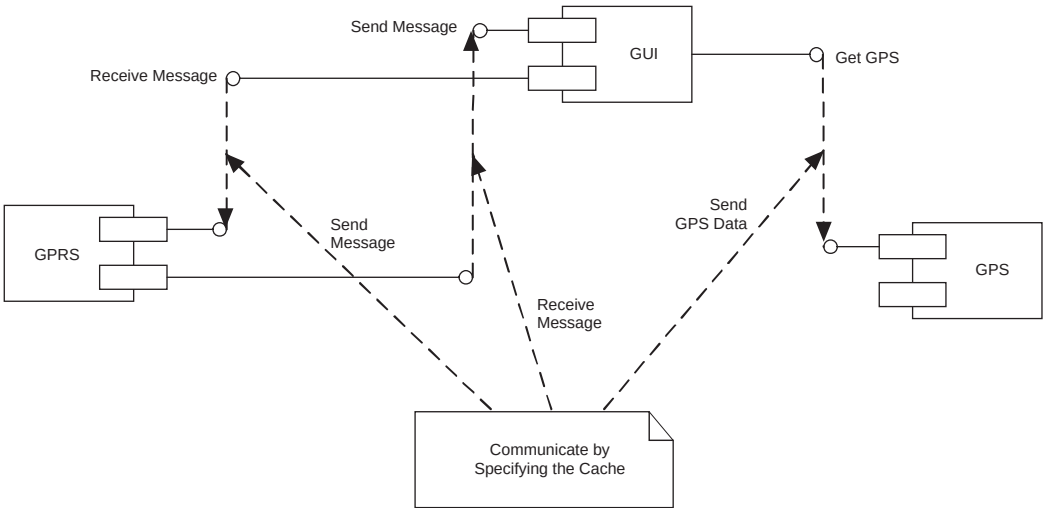
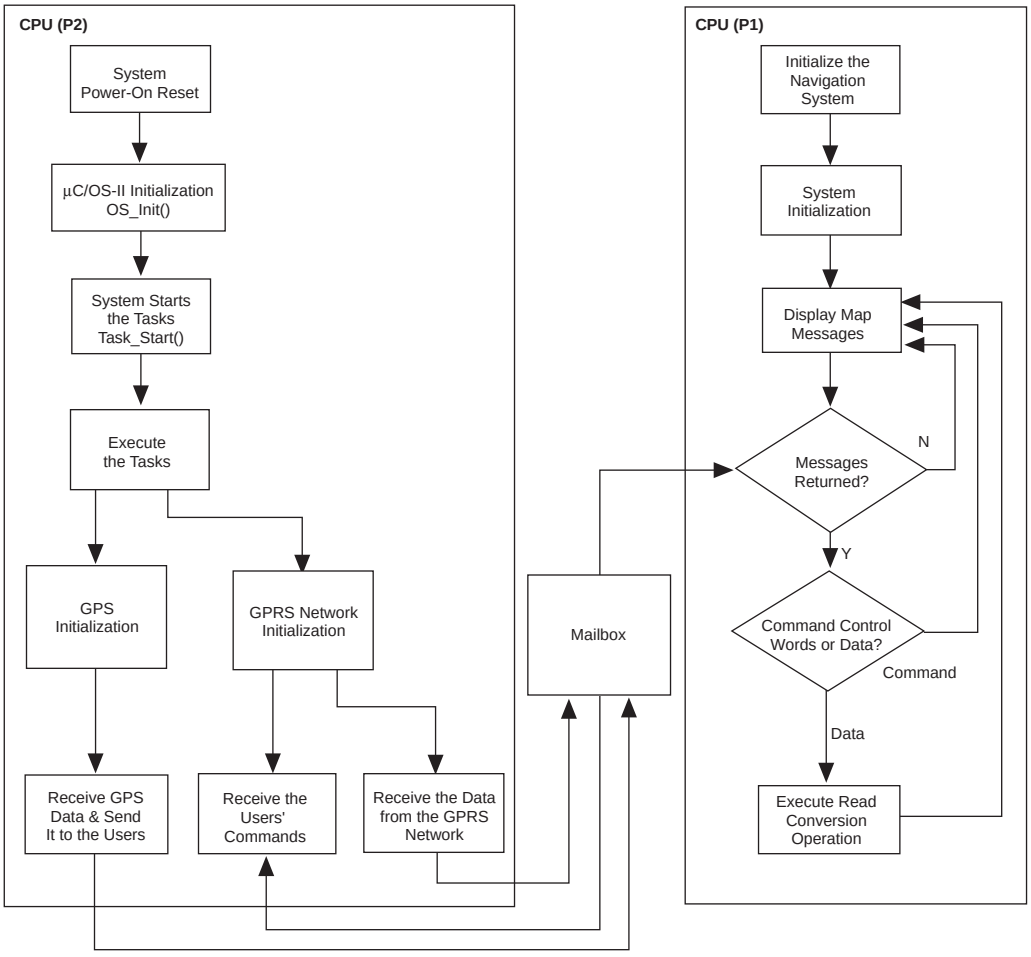


Figure 6 shows the software flow.

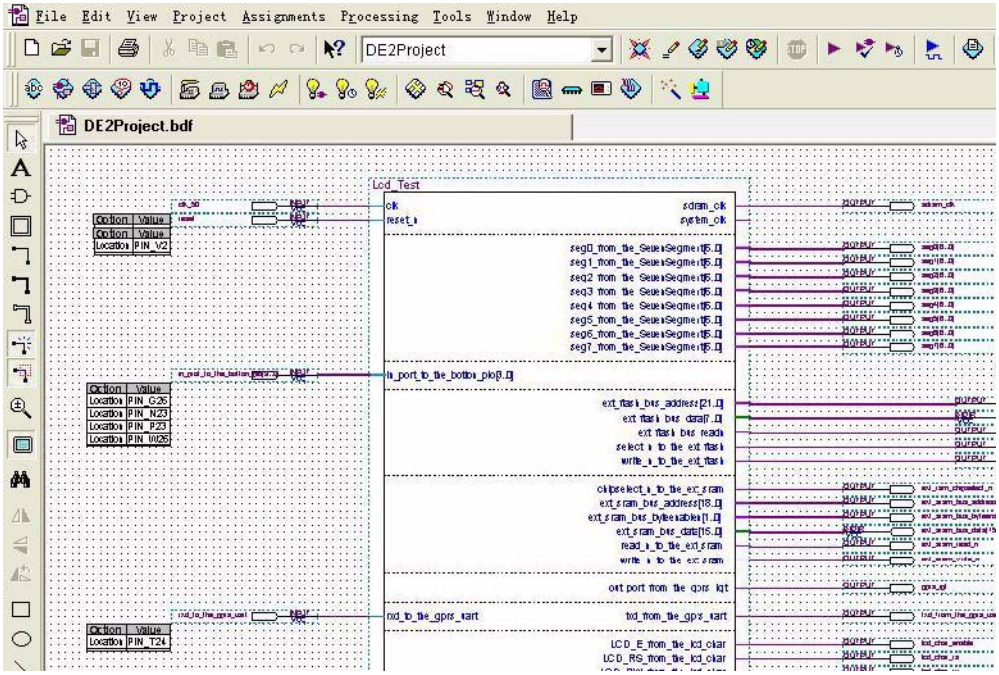
Figure 6. Software Flow Design



Design Methodology

Our methodology was to use an FPGA to construct our system architecture, including the LCD interface, touch screen interface, serial interface, etc. See Figure 7. The following sections describe our design in more detail.

Figure 7. System Architecture



Nios II Dual-Core Processor Design

As we developed our design, we determined that implementing µC/GUI was complex, that we needed heterogeneous map data processing, and that µc/OS-II required a huge stack when deploying the GUI. These factors made for a poor final product, so we decided not to use µc/OS-II for the whole project. Instead, we operated µC/GUI without the OS and the GPS and GPRS modules with the OS. To implement it, we chose a dual-core design.

The dual-core system fully uses the Nios II processor's features, and has the following arrangement:

- First CPU:
  - Perform human-machine interaction without the OS.
  - Operate µC/GUI and the application program.
- Second CPU:
  - Set up the embedded real-time µC/OS-II OS.
  - Receive the front-end GPS data through the UART1 serial interface.
  - Communicate with the GPRS module through the UART2 (self-defined expansion) serial interface.

The two Nios II processors communicate through a mailbox. The dual-core resistive communication mechanism did not operate well, so we created a mailbox mechanism to allow resistive communication between the two cores.

A mutual exclusion object (mutex) and mailbox allow the two cores to share memory, which is 2 x 32 bits (see Table 2). We set the value to 0x0000, set the reset to 1, and enable the mutex. The mailbox allows the cores to communicate. The mutex resides in the mailbox and waits to be modified by a CPU. The following code shows the structure of the shared memory and mailbox.

```
typedef struct {
    alt_u8     Flag; // avoid repeated reading and writing of the data
    alt_u32    Data_Length;  // data length
    alt_u8  *  Data_Buf; // data cache
}SHARE_MEMERY;

typedef struct {
    void           * Mutex_Base;
    SHARE_MEMERY   * Share_Memery;
}MAIL_BOX;
```

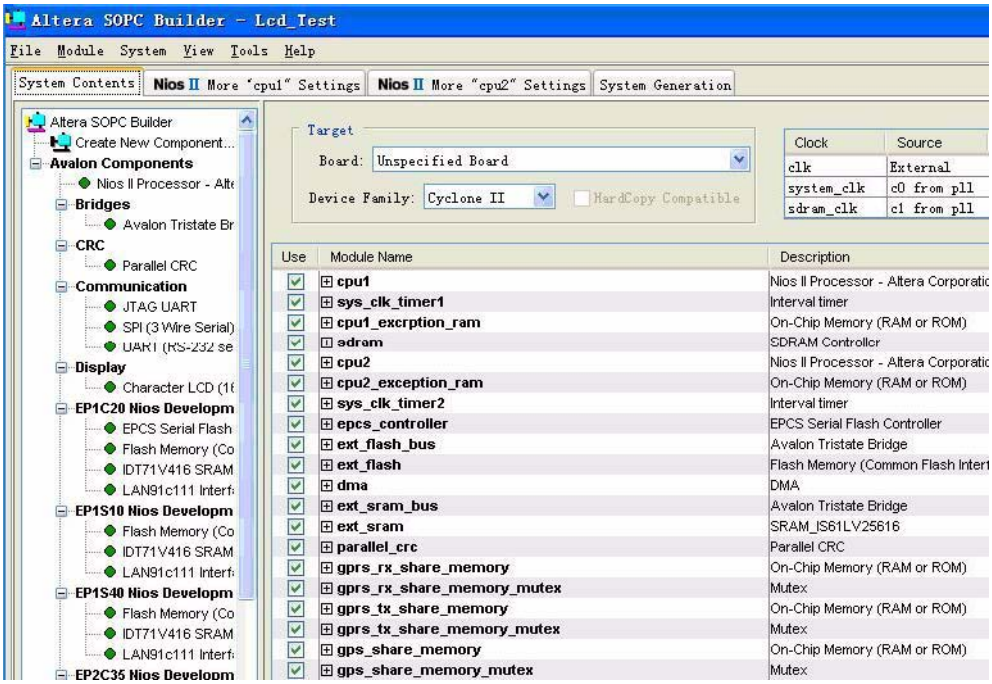
Table 2 shows the memory structure.

Table 2. Shared Memory

Offset	Register Name	Read/write	Bit Descriptor		
			31 … 16	15 … 1	0
0	mutex	RW	OWNER	VALUE	
1	reset	RW	N/A	N/A	RESET

Figure 8 shows the hardware system, which is generated by adding intellectual property cores in SOPC Builder.

Figure 8. SOPC Builder Hardware System



LCD and Touch Screen

The Hitachi SX19V001-Z2A LCD and touch screen have 640 x 480 resolution and an eight-color screen. The touch screen is 4-wire resistive. The LCD controller is the Epson SED13503. The touch screen's drive module is the analog-to-digital (A/D) sampling converter MXB7843. It connects to the 4-wire resistive touch screen, and converts the touch screen voltage value into a value we can work with more easily (the conversion is 12 bits and 1/4,096 of the X or Y direction). The MXB7843 devices sends the controller the X, Y values that have gone through A/D conversion, which is not practical without translation. The touch screen's resistance value fluctuates in nearly a line, which means we can use the following translation formulas:

X coordinate = 640 x (X - X1) / (X2 - X1)

Y coordinate = 480 x (Y - Y1) / (Y2 - Y1)

Where the LCD's resolution is 640 x 480, the voltage on LCD's upper left corner is X1 and Y1, the voltage on its lower right corner is X2 and Y2, and the voltage on any point of the screen is X and Y.

We designed the LCD touch screen drive circuit, which connects to the self-defined expanded bus and SPI interface, i.e., the address, data, and control signals. All signals are implemented using the two universal input and output interfaces (JP1 and JP2) on the Altera® Development and Education (DE2) board.

GPRS Module

We use the Siemens MC35I device as the GPRS module. This module has passed radio and telecommunications terminal equipment (R&TTE) and Global Certification Forum (GCF) authentication. It supports end-to-end and end-to-user communication, SMS and GPRS-like data transmission and voice calls, and the GSM07.07 protocol. It also provides a variety of interfaces, including a subscriber identity module (SIM) card interface, power interface, standard RS-232 interface, etc.

Because the MC35I communication module does not support parity, we used serial communication with a 115,200 bps baud rate, 8 data bits, 1 stop bit, no parity bit, and without streaming control. This arrangement occupies minimal CPU time.

Compared to other wireless communication technologies, GPRS has real-time operation and stability. It also has the following features:

- *Always on-line*—As long as GPRS service is activated, it remains on-line, which is similar to a wireless network.
- *Billing by use*—Although it is always on-line, the user does not have to worry about the expense. The system only triggers billing when communication traffic is generated. The utilization-based billing mechanism is more reasonable and logical than other methods.
- *Quick login*—The packet service is much faster than a dial-up connection.
- *High-speed transmission*—Theoretically, GPRS's maximum transmission speed is 171.2 Kbps. The GPRS systems currently being used support transmission speeds of about 40 Kbps.

GPS Module

The GPS module uses the GS1100 device. Its receiver has high sensitivity (up to -153 dBm), features low power consumption, has a built-in passive antenna, conforms to the National Marine Electronics Association (NMEA) v3.0 protocol, and transfers data through the RS-232 interface.

CRC Check Drive

Based on Altera's cyclic redundancy code (CRC) check, we designed a new CRC check in hardware. We implemented a large network data check and accelerated the checking speed.

Hardware System Description

This section describes our system's hardware design.

µC/OS-II

µC/OS-II is the basis of the GPRS communication module. It allows communication between tasks in the communication module and performs a variety of tasks, such as `Task_start` (main control tasks), `Task_GPS_Send` (receives GPS data from the serial interface and sends GPS messages to the GUI), `Task_GPRS_Send` (sends the data received from the background to the GUI), and `Task_GPRS_TCP_Send` (sends front-end service inquiries to the background console).

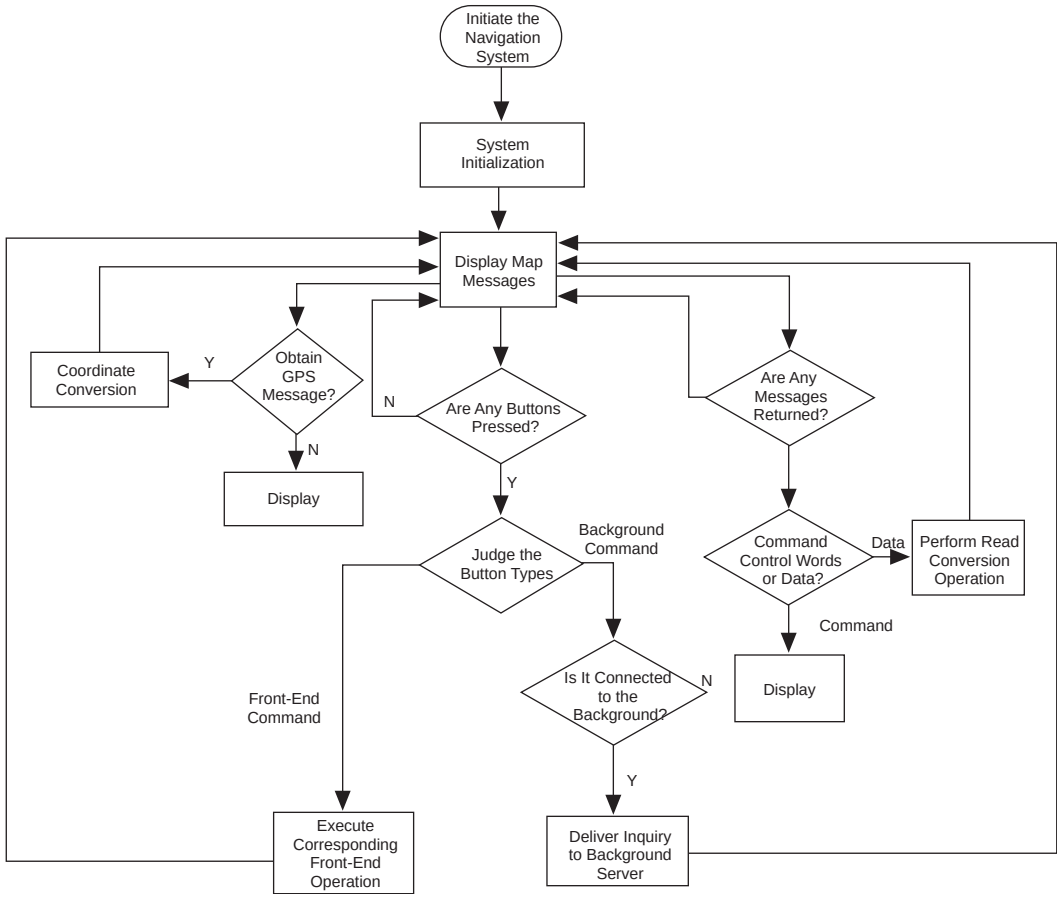
µC/GUI and Application

µC/GUI provides interface elements (form and controls), drawing functions, excellent color management, Chinese character support, and a hardware driver. We developed our application based on the GUI. We used a compressed bitmap for the intelligent traffic navigation system interface and the storage format. See Figure 9.

Figure 9. Compressed Bitmap for Navigation Interface

Figure 10 shows the application software flow chart.

Figure 10. Application Software Flow Chart



Notes to Figure 10:

(1) Front-end command means all of the commands except the surrounding messages on the interface.
(2) Background command means the surrounding messages.

Our design does not use µC/GUI and µC/OS-II together for several reasons:

- Implementing µC/GUI is very complex.
- Our design requires heterogeneous map data processing.
- µC/OS-II requires a huge stack when the GUI is implemented at the same time.
- µC/GUI and µC/OS-II have little connection.

To implement the GUI and software application we defined several files:

- *GUIConf.h*—Describes the size of µC/GUI function module and dynamic storage (used for memory devices and window object), the default font setup, and other basic GUI control definitions.

- *LCDConf.h*—Provides the LCD parameter control files such as LCD size, controller types, bus width, color selection, etc.

- *GUI/CORE/LCD_ConfDefaults.h*—Describes all default configuration items, such as the LCD numbers, controller numbers, color palette, the screen's reverse setup, etc.

- *GUITouchConf.h*—Configures the touch screen by compiling the following functions according to the touch screen and the control chip (we need not the X, Y data in the software, so these functions are empty structures):

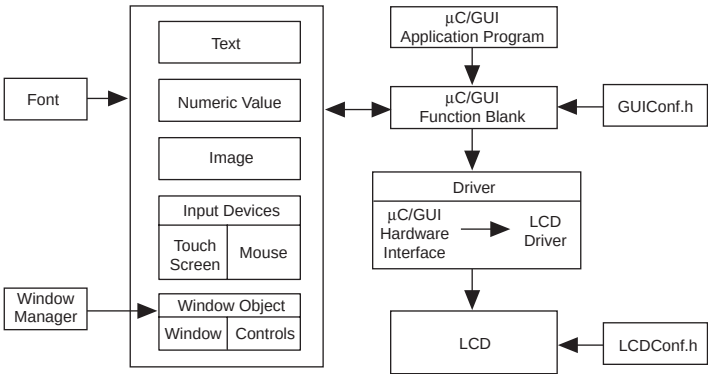
```
void TOUCH_X_ActivateX (void)?// Ready to measure the data on X-axis
void TOUCH_X_ActivateY (void)?// Ready to measure the data on Y-axis
```

The following functions are called by `GUI_TOUCH_Exec()`:

```
int TOUCH_X_MeasureX(void); // Return the value of X according to AD
                            // conversion result
int TOUCH_X_MeasureY(void); // Return the value of Y according to AD
                            // conversion result
```

Figure 11 shows the µC/GUI software structure.

Figure 11. µC/GUI Software Structure



Using the built-in µC/GU function, `memc_driver`, we convert the cache reading and writing into that of the memory (physical address), which allows the LCD to display the images correctly.

The run-length encoding (RLE) algorithm goes through a scan line and replaces adjacent pixels with the same color value based on the number of occurances of the color and the color value of the pixels. For example, aaabccccccddeee is replaced by 3a1b6c2d3e. Using the RLE algorithm, fully compressing the µC/GUI interface requires a large memory and a lot of CPU time. Our system does not need full compression and our map has consecutive white pixels as the dominant color. Therefore, we used an RLE-like algorithm (only compressing the white pixels) to accelerate the compression rate while slightly decreasing the compression ratio. This technique, along with using a partial compression strategy (we only compress the part that is displayed on the screen), means that we do not need as much memory.

The DE2 board's flash device is 4 Mbytes and our map can be as much as 4.1 Mbytes. So we have to compress the map if we want to use it. With the RLE-like algorithm, we compressed our map to 2.45 Mbytes, which saves a significant amount of flash memory.

GPRS Communication Module

This section describes the methods the GPRS uses for communication.

UART Serial Driver

To avoid CPU hangs caused by the Nios II UART driver and to help determine the package's start and end points, we compiled a UART driver suited for our system that operates in real time and is easy to use. The driver's data structure shown below:

```
typedef struct {
   alt_u32             BASE;
   alt_u32             IRQ_ID;
   alt_u8              Rx_Buf [ Uart_Buf_Size ];
   alt_u32             Rx_Bytes;
   alt_u32             In_Ptr;
   alt_u8              Rx_Star_Char;
   alt_u8              Rx_End_Char;
   alt_u16             Same_Flage;
   Data_Buf_Queue  *   Data_Queue;
   OS_EVENT        *   Sem;
   OS_EVENT        *   Rx_Sem;
}UART;
```

Controlling the MC35I Module's AT Instruction

The GSM07.07 standard defines complete AT instructions. Our system only uses instructions to control a short message, phone number directory, and basic AT instructions expanded by GPRS. AT instructions have the form AT+XXXXXX. The system sends the AT instructions' ASCII codes to the GPRS communication module, then delivers the \n ASCII code to confirm the command. After executing the command, the GPRS communication module returns the result, which is \r\nOK\r\n for success and \r\nERROR\r\n for failure. Table 3 shows the basic AT instructions we used.

Table 3. AT Instructions

Instruction	Function Description
AT	If an OK is returned, the GPRS module supports the AT instruction and operation is normal.
AT+CREG	Network registration.
AT+IPR	Set data terminal device velocity.
AT+CGDCONT	Define PDP context.
AT+CGACT	PDP context activation.
AT+CGQMIN	Service quality introduction (as small as acceptable).
AT&W	Reserve all software modification of the module.

Custom Instructions

The communication module uses the following custom instructions to implement the big/small end conversion.

```
void IP_Assemble( IP  *  IP_Packet ) {
   IP_Packet->Total_Len  =  ( ALT_CI_ENDIAN ( IP_Packet->Total_Len ) >> 16 );
   IP_Packet->ID         =  ( ALT_CI_ENDIAN ( IP_Packet->ID ) >> 16 );
   IP_Packet->Offset     =  ( ALT_CI_ENDIAN ( IP_Packet->Offset ) >> 16 );
   IP_Packet->Ckecksum   =  ( ALT_CI_ENDIAN ( IP_Packet->Ckecksum ) >> 16 );
   IP_Packet->Src_IP     =  ALT_CI_ENDIAN ( IP_Packet->Src_IP );
   IP_Packet->Dst_IP     =  ALT_CI_ENDIAN ( IP_Packet->Dst_IP );

   memcpy ( &PPP_Packet_Buf [ PPP_Packet_Buf_Length ], IP_Packet,
      sizeof ( IP ) - 8 );
}
```

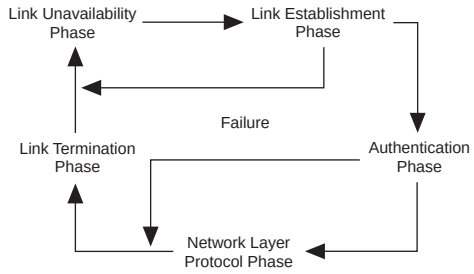
Point-to-Point Protocol (PPP)

In this system, the µC/OS-II OS controls the GPRS module to access the network, implementing point-to-point communication. PPP provides a standard method to encapsulate a multi-protocol message on

the point-to-point link. It supports the dynamic distribution and management of IP addresses, the transmission of a synchronous (delivery of bit-oriented synchronous data module) or asynchronous (start bit + data bit + parity bit + stop bit) physical layer, network layer protocol reuse, link configuration, quality detection, and error correction, and can negotiate multiple parameters.

PPP has three parts: the link control protocol (LCP), network control protocol (NCP), and expanded protocols, such as the multi-link protocol. To establish PPP communication, the system first sends an LCP package to each side of the PPP connection to configure and test the data connection. After the connection is established, the corresponding entity is confirmed. Then, the system sends an NCP package to select one or more network layer protocols for configuration. Once the selected network layer protocol is configured, the package is delivered on the link. The link remains in a configurable state unless the LCP and NCP packages terminate the connection or other unexpected events occur (e.g., the non-activity clock is fully occupied by counting or there is interference). Figure 12 shows the PPP link phase transition.

Figure 12. Link Phase Transition



TCP/IP

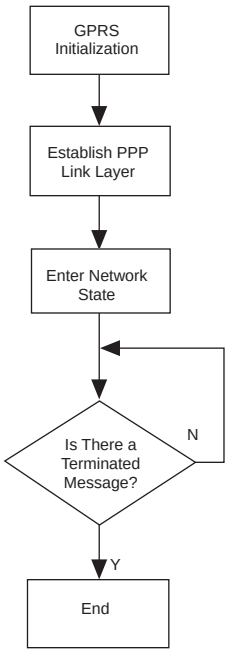
The system must deliver collected data to the server immediately through the Internet. The delivery must be in real time and fast, but have a small amount of data. Because of these requirements and the functions implemented using TCP/IP, we simplified the subprotocols as shown in Table 4.

Table 4. Simplified TCP/IP Subprotocols

Layer	Implemented Protocol
Application layer	Subscriber self-defined
Transmission layer	TCP/UDP
Network layer	IP
Link layer	PPP

Figure 13 shows the GPRS communication module flow chart according to these protocols. We implemented PPP, TCP/IP, and UDP ourselves.

Figure 13. GPRS Communication Module Flow Chart



GPS Positioning

The vehicle navigation system is part of the intelligent traffic system. Its built-in GPS antenna receives data from at least 3 out of 24 GPS satellites orbiting the Earth. With help from the navigation system's e-map, it converts the latitude and longitude position coordinates measured by the GPS satellite signals, which confirms the vehicle's position on the map.

Client-Server Application

We created the server-side software using Microsoft SQL2000 and Microsoft Visual Studio .NET 2003. The database contains hotels, shopping malls, hospitals, railways and bus stations, gas stations, roads, etc. The .NET system mainly deals with interface design, algorithm design, and data processing.

The client-server mode has the following advantages:

- It reduces the front-end burden and costs by placing the information in the background.
- It facilitates software function expansion. The driver can quickly download another city's map (such as Xi'an).
- It requires real-time operation. The background server can immediately send the road conditions to the driver.
- The navigation system's powerful background server provides real-time geographical information as well as traffic jam conditions, helping the driver learn about road conditions and to guide traffic at rush hour.

Figure 14 shows the background server's login interface.

Figure 14. Background Server Login Interface

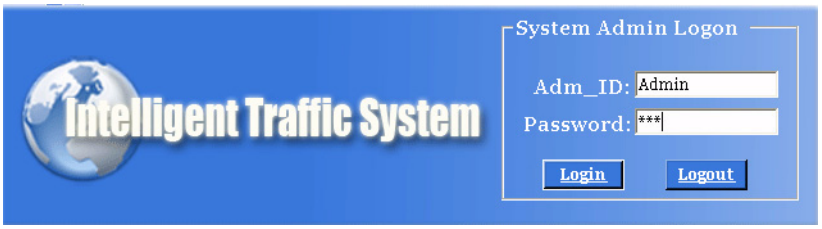
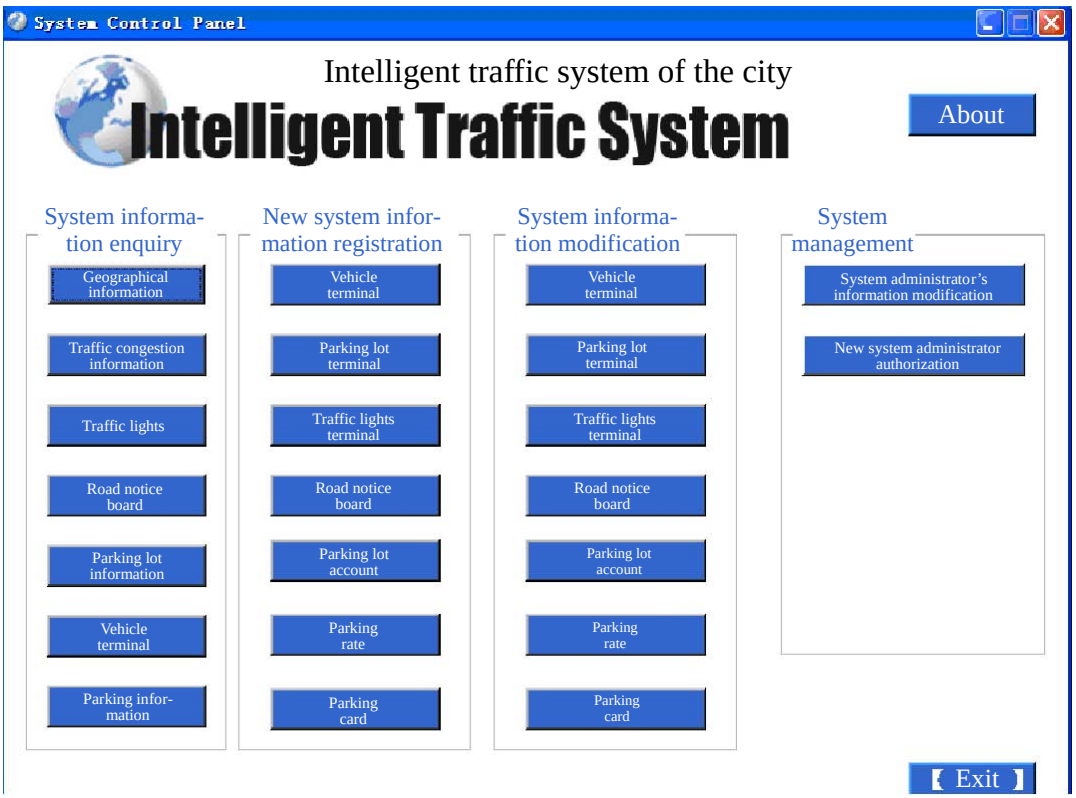


Figure 15 shows the interface operation.

Figure 15. Interface Operation



Traffic Congestion Algorithm

This algorithm is used in server terminals. Using client-server advantages, the vehicle terminal can be used as a data collection terminal for the traffic authority. As long as the traffic authority has the system (e.g., installed in a bus or taxi), it will be informed of road congestion according to the information sent

from the vehicle terminal. (It is also available if the user does not send the information.) The algorithm is defined as:

If R << R0 and N is bigger than a specified number (to reduce the possibility that parking affects the result), the road segment is considerd to be congested. Congestion increases as R decreases.

where:

- N is the number of vehicles that are installed with this device and running on the road S.
- R is the average vehicle speed (the specified time is longer than the duration of a red light to avoid errors).
- R0 is the road's speed limit.

Design Features

Our design has the following features:

- *Uses two Nios II processors*—Using two processors solves software design problem and improves system performance. Using an FPGA to implement the two processors reduces hardware costs and improves performance.
- *Custom peripherals and instructions*—The LCD, touch screen GPS and GPRS, all as custom peripheral processing, use a custom bus and SPI (the 40-pin expansion headers on the Altera DE2 board) to complete the interface design. We use custom instructions to implement the big/small end conversion of the GPRS communications module.
- *Client-server mode*—The server can provide better services for drivers using client-server mode.
- *Traffic jam algorithm and best path*—With the traffic jam algorithm described previously, traffic authorities can learn traffic conditions, helping them guide traffic and send traffic updates to drivers who can then choose the best route.
- *Integrating µC/GUI and µC/OS-II in the embedded system*—µC/GUI is the design platform of the whole navigation module, which provides various interface elements (windows and controls), good color management, Chinese character support, and low-level driver support for the hardware. We used it to compile applications and provide a friendly user interface. We used µC/OS-II, which is effective and reliable, to deploy system tasks.
- *Using TCP/IP and PPP in the embedded system*—Applying TCP/IP and PPP to the system did not require memory management and allowed transmission of small volumes of data.
- *GPRS*—To send real-time road information to drivers and to facilitate communication between drivers and the control center, the system establishes an external GPRS module and connects with the Nios II processor in the FPGA using the serial port. Drivers can directly send service requests to the back-end monitoring center and receive information. Meanwhile, the back-end learns traffic conditions in real-time by collecting the road information.
- *GPS*—We used a GPS to implement real-time, accurate vehicle positioning and to provide the necessary data for the traffic jam algorithm.

- *Touch screen LCD*—Touch operations are easy and the LCD is beautiful, offering a user-friendly interface.

- *Collaborative hardware/software development*—Because the hardware and software in the SOPC Builder can be clipped, we can jointly develop the hardware and software during system development. Hardware and software development begin and end almost simultaneously, saving time. Additionally, we accelerated the product launch and can support a longer product life cycle. If we need to modify some definitions during development, we can simply generate a new Nios II core, which does not impact other peripherals or Nios II programs.

Conclusion

In our two-month project, we successfully designed the Nios II-based multiple-core intelligent vehicle terminal, improving our knowledge of the Nios II embedded system. By meeting specific requirements, the multiple-core processor solution provides higher performance without adding more hardware. Using dual-core processor technology is another way to enhance a processor's performance. Because the processor's real performance is the sum of the instructions that processes per clock cycle, adding another processor doubles the number of instructions executed per clock cycle.

The system design is centered on one FPGA. Using a range of Nios II functions and features, the control and data processing are managed by two Nios II processors implemented in an FPGA, demonstrating the high integration of system-on-a-programmable-chip (SOPC) solutions. Designing the system with SOPC tools helped us to streamline the hardware and software system, disconnecting the software development from the hardware development. Moreover, by combining multiple processors and selecting suitable peripherals, memories, and I/O interfaces, we could build a customized embedded system with the lowest price, simplest design, highest performance, and lowest risk.

The design uses the new Nios II Integrated Development Environment (IDE), which, compared with the original environment, has simpler register operation and a more convenient operating and debugging environment. The original development tool was difficult to download and debug because it did not have a graphical compiling and debugging environment and download speeds were slow. The new Nios II IDE downloads and debugs through the JTAG-UART with a Windows-style interface.

The competition showed us how to develop a project and helped us learn a new design concept: SOPC. We believe cooperation is vital to the success of such an arduous task with many sophisticated technologies. Without a joint effort and team spirit, we would never have finished the design. We will always have good memories of the experience.

Due to limited time and resources, the system is unsatisfactory in some areas. The GPS cannot receive GPS satellite signals indoors, so after starting the indoor system normally, the GPS shows the GPS information that was saved when the system was last closed.

Due to limited storage space and data, we only made a map for Xi'an. Our design is missing data for other cities, e.g., Shanghai, and information on relevant places.

Appendix: GPRS Communications Testing Data

Refer to the PDF of this paper on the Altera web site at http://www.altera.com for a detailed description of the GPRS communications testing data.

Appendix: GPRS Communications Testing Data:

```
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

ATE0

OK
AT

OK
AT+CREG?

+CREG: 0,1
AT+IPR=115200

OK
AT+CGDCONT=1,"ip","cmnet",,0,0

OK
AT&W

OK
AT+CGACT=1

OK
AT+CGATT=1

OK
AT+CGQMIN=1,0,0,3,0,0

OK

*****************************************************
**- < GPRS Module Initialization is Successfull > -**
*****************************************************
ATD*99**1*1#

CONNECT

*****************************************************
**-- < Connection GPRS Network is  Successfull > --**
*****************************************************

Rx_Data:_____
```

_____
FCS is OK!
0x7e 0xff 0x 3 0xc0 0x21 0x 1 0x 3 0x 0 0x19 0x 2 0x 6 0x 0 0x a 0x 0 0x 0 0x 7
0x 2 0x 8 0x 2 0x 5 0x 6 0x53 0x8e 0xb6 0xef 0x 3 0x 5 0xc2 0x23 0x 5 0xe1 0x5e
0x7e
Rx LCP Req !

Tx LCP Req :
Tx_Data:_____
_____
FCS is OK!
0x7e 0xff 0x7d 0x23 0xc0 0x21 0x7d 0x21 0x7d 0x21 0x7d 0x20 0x7d 0x32 0x7d 0x22
0x7d 0x26 0x7d 0x20 0x7d 0x2a 0x7d 0x20 0x7d 0x20 0x7d 0x27 0x7d 0x22 0x7d 0x28
0x7d 0x22 0x7d 0x23 0x7d 0x24 0xc0 0x23 0x6c 0x7d 0x30 0x7e

Rx_Data:_____
_____
FCS is OK!
0x7e 0xff 0x 3 0xc0 0x21 0x 2 0x 1 0x 0 0x12 0x 2 0x 6 0x 0 0x a 0x 0 0x 0 0x 7
0x 2 0x 8 0x 2 0x 3 0x 4 0xc0 0x23 0x54 0x11 0x7e
Rx LCP ACK !

Rx_Data:_____
_____
FCS is OK!
0x7e 0xff 0x 3 0xc0 0x21 0x 1 0x 3 0x 0 0x18 0x 2 0x 6 0x 0 0x a 0x 0 0x 0 0x 7
0x 2 0x 8 0x 2 0x 5 0x 6 0x53 0x8e 0xb6 0xef 0x 3 0x 4 0xc0 0x23 0x1e 0x95 0x7e
Rx LCP Req !

Tx LCP ACK :
Tx_Data:_____
_____
FCS is OK!
0x7e 0xff 0x7d 0x23 0xc0 0x21 0x7d 0x22 0x7d 0x23 0x7d 0x20 0x7d 0x38 0x7d 0x22
0x7d 0x26 0x7d 0x20 0x7d 0x2a 0x7d 0x20 0x7d 0x20 0x7d 0x27 0x7d 0x22 0x7d 0x28
0x7d 0x22 0x7d 0x25 0x7d 0x26 0x53 0x8e 0xb6 0xef 0x7d 0x23 0x7d 0x24 0xc0 0x23
0xd2 0x78 0x7e

Tx PAP Req: :
Tx_Data:_____
_____
FCS is OK!
0x7e 0xff 0x7d 0x23 0xc0 0x23 0x7d 0x21 0x7d 0x22 0x7d 0x20 0x7d 0x26 0x7d 0x20
0x7d 0x20 0x4d 0x34 0x7e

Rx_Data:_____
_____
FCS is OK!
0x7e 0xc0 0x23 0x 2 0x 2 0x 0 0x 5 0x 0 0x30 0x15 0x7e
Rx PAP ACK !

Tx IPCP Req:
Tx_Data:_____
_____
FCS is OK!
0x7e 0xff 0x7d 0x23 0x80 0x21 0x7d 0x21 0x7d 0x21 0x7d 0x20 0x7d 0x36 0x7d 0x23 0x7d 0x26 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x81 0x7d 0x26 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x83 0x7d 0x26 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x6e 0xdb 0x7e

Rx_Data:_____
_____
FCS is OK!
0x7e 0x80 0x21 0x 1 0x 1 0x 0 0x a 0x 3 0x 6 0xc0 0xa8 0xfe 0xfe 0x48 0xcc 0x7e
Rx IPCP Req !

Tx IPCP ACK:
Tx_Data:_____
_____
FCS is OK!
0x7e 0xff 0x7d 0x23 0x80 0x21 0x7d 0x22 0x7d 0x21 0x7d 0x20 0x7d 0x2a 0x7d 0x23 0x7d 0x26 0xc0 0xa8 0xfe 0xfe 0x5f 0x56 0x7e

Tx IPCP Req:
Tx_Data:_____
_____
FCS is OK!
0x7e 0xff 0x7d 0x23 0x80 0x21 0x7d 0x21 0x7d 0x22 0x7d 0x20 0x7d 0x36 0x7d 0x23 0x7d 0x26 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x81 0x7d 0x26 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x83 0x7d 0x26 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0xda 0x82 0x7e

Rx_Data:_____
_____
FCS is OK!
0x7e 0x80 0x21 0x 3 0x 2 0x 0 0x16 0x 3 0x 6 0x a 0xb6 0x 9 0x b 0x81 0x 6 0xd3 0x89 0x82 0x 3 0x83 0x 6 0xd3 0x89 0x82 0x13 0xc3 0xcd 0x7e
Rx IPCP NCK !

Get IP Adress:

_____GPRS Network Config Information_____

This is My IP Adress: 10.182.9.11.
This is My DNS 1 Adress: 211.137.130.3.
This is My DNS 2 Adress: 211.137.130.19.
_____

Tx_Data:_____
_____
FCS is OK!
0x7e 0xff 0x7d 0x23 0x80 0x21 0x7d 0x21 0x7d 0x23 0x7d 0x20 0x7d 0x36 0x7d 0x23 0x7d 0x26 0x7d 0x2a 0xb6 0x7d 0x29 0x7d 0x2b 0x81 0x7d 0x26 0xd3 0x89 0x82 0x7d 0x23 0x83 0x7d 0x26 0xd3 0x89 0x82 0x7d 0x33 0x64 0x30 0x7e

Rx_Data:_____
_____
FCS is OK!
0x7e 0x80 0x21 0x 2 0x 3 0x 0 0x16 0x 3 0x 6 0x a 0xb6 0x 9 0x b 0x81 0x 6 0xd3 0x89 0x82 0x 3 0x83 0x 6 0xd3 0x89 0x82 0x13 0xf2 0x53 0x7e

_____GPRS Network Config Information_____

This is My IP Adress: 10.182.9.11.
This is My DNS 1 Adress: 211.137.130.3.
This is My DNS 2 Adress: 211.137.130.19.
_____

****
-- < PPP Link Config is Successfully! > --
****

Tx_Data (IP_Packet):
Tx_Data:_____
_____
FCS is OK!
0x7e 0x21 0x45 0x7d 0x20 0x7d 0x20 0x26 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0xff 0x7d 0x31 0x5c 0x54 0x7d 0x2a 0xb6 0x7d 0x29 0x7d 0x2b 0xca 0x75 0x81 0x3c 0x7d 0x20 0x64 0x7d 0x24 0x7d 0x38 0x7d 0x20 0x7d 0x32 0x91 0xd6 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x30 0x62 0x8d 0x7e

Tx_Data (IP_Packet):
Tx_Data:_____
_____
FCS is OK!
0x7e 0x21 0x45 0x7d 0x20 0x7d 0x20 0x26 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20
0xff 0x7d 0x31 0x5c 0x54 0x7d 0x2a 0xb6 0x7d 0x29 0x7d 0x2b 0xca 0x75 0x81 0x3c
0x7d 0x20 0x64 0x7d 0x24 0x7d 0x38 0x7d 0x20 0x7d 0x32 0x91 0xd6 0x7d 0x20 0x7d
0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x31 0x32 0x33
0x34 0x35 0x36 0x37 0x38 0x39 0x30 0x62 0x8d 0x7e

Tx_Data (IP_Packet):
Tx_Data:_____
_____
FCS is OK!
0x7e 0x21 0x45 0x7d 0x20 0x7d 0x20 0x26 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20
0xff 0x7d 0x31 0x5c 0x54 0x7d 0x2a 0xb6 0x7d 0x29 0x7d 0x2b 0xca 0x75 0x81 0x3c
0x7d 0x20 0x64 0x7d 0x24 0x7d 0x38 0x7d 0x20 0x7d 0x32 0x91 0xd6 0x7d 0x20 0x7d
0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x31 0x32 0x33
0x34 0x35 0x36 0x37 0x38 0x39 0x30 0x62 0x8d 0x7e

TCP Establishment is runningh

Tx_Data (IP_Packet):
Tx_Data:_____
_____
FCS is OK!
0x7e 0x21 0x45 0x7d 0x20 0x7d 0x20 0x28 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20
0xff 0x7d 0x26 0x5c 0x5d 0x7d 0x2a 0xb6 0x7d 0x29 0x7d 0x2b 0xca 0x75 0x81 0x3c
0x7d 0x20 0x50 0x7d 0x24 0x7d 0x38 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d
0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x50 0x7d 0x22 0x7d 0x28 0x7d 0x20 0x44 0x7d
0x28 0x7d 0x20 0x7d 0x20 0x7d 0x2b 0x7d 0x37 0x7e

Rx_Data:_____
_____
FCS is OK!
0x7e 0x21 0x45 0x 0 0x 0 0x2c 0x13 0x2f 0x40 0x 0 0x2f 0x 6 0xd9 0x2a 0xca 0x75
0x81 0x3c 0x a 0xb6 0x 9 0x b 0x 4 0x18 0x 0 0x50 0x32 0xe2 0x56 0xe7 0x 0 0x 0
0x 0 0x 1 0x60 0x12 0xff 0xff 0xaa 0x71 0x 0 0x 0 0x 2 0x 4 0x 5 0xb4 0x20 0xe0
0x7e

-----TCP Checksum is Eroor!

Tx_Data (IP_Packet):
Tx_Data:_____
_____
FCS is OK!
0x7e 0x21 0x45 0x7d 0x20 0x7d 0x20 0x28 0x7d 0x33 0x2f 0x40 0x7d 0x20 0x2f 0x7d 0x26 0xd9 0x2e 0x7d 0x2a 0xb6 0x7d 0x29 0x7d 0x2b 0xca 0x75 0x81 0x3c 0x7d 0x20 0x50 0x7d 0x24 0x7d 0x38 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x21 0x32 0xe2 0x56 0xe8 0x50 0x7d 0x30 0x7d 0x28 0x7d 0x20 0xba 0x2e 0x7d 0x20 0x7d 0x20 0x9c 0xea 0x7e

Tx_Data (IP_Packet):
Tx_Data:_____
_____
FCS is OK!
0x7e 0x21 0x45 0x7d 0x20 0x7d 0x20 0x32 0x7d 0x33 0x2f 0x40 0x7d 0x20 0x2f 0x7d 0x26 0xd9 0x24 0x7d 0x2a 0xb6 0x7d 0x29 0x7d 0x2b 0xca 0x75 0x81 0x3c 0x7d 0x20 0x50 0x7d 0x24 0x7d 0x38 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x21 0x32 0xe2 0x56 0xe8 0x50 0x7d 0x38 0x7d 0x28 0x7d 0x20 0xb0 0x7d 0x37 0x7d 0x20 0x7d 0x20 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x30 0x3d 0x7d 0x27 0x7e

Rx_Data:_____
_____
FCS is OK!
0x7e 0x21 0x45 0x 0 0x 0 0x28 0x13 0x44 0x40 0x 0 0x30 0x 6 0xd8 0x19 0xca 0x75 0x81 0x3c 0x a 0xb6 0x 9 0x b 0x 4 0x18 0x 0 0x50 0x32 0xe2 0x56 0xe8 0x 0 0x 0 0x 0 0x b 0x50 0x10 0xff 0xf5 0xc2 0x2e 0x 0 0x 0 0x 6 0xb7 0x7e

Rx_Data:_____
_____
FCS is OK!
0x7e 0x21 0x45 0x 0 0x 0 0x2e 0x13 0x57 0x40 0x 0 0x30 0x 6 0xd8 0x 0 0xca 0x75 0x81 0x3c 0x a 0xb6 0x 9 0x b 0x 4 0x18 0x 0 0x50 0x32 0xe2 0x56 0xe8 0x 0 0x 0 0x 0 0x b 0x50 0x18 0xff 0xf5 0x28 0x84 0x 0 0x 0 0x31 0x32 0x33 0x34 0x35 0x36 0x50 0xd5 0x7e

----------Receive Data :
123456
Tx_Data (IP_Packet):
Tx_Data:_____
_____
FCS is OK!
0x7e 0x21 0x45 0x7d 0x20 0x7d 0x20 0x28 0x7d 0x33 0x57 0x40 0x7d 0x20 0x30 0x7d 0x26 0xd8 0x7d 0x26 0x7d 0x2a 0xb6 0x7d 0x29 0x7d 0x2b 0xca 0x75 0x81 0x3c 0x7d 0x20 0x50 0x7d 0x24 0x7d 0x38 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x2b 0x32 0xe2

0x56 0xee 0x50 0x7d 0x30 0x7d 0x28 0x7d 0x20 0xba 0x7d 0x3e 0x7d 0x20 0x7d 0x20 0x8d 0x7d 0x22 0x7e

The Data Queue is Empty
The Packet is Retransmission

Tx_Data (IP_Packet):
Tx_Data:_____
_____
FCS is OK!
0x7e 0x21 0x45 0x7d 0x20 0x7d 0x20 0x2b 0x7d 0x33 0x57 0x40 0x7d 0x20 0x30 0x7d 0x26 0xd8 0x7d 0x23 0x7d 0x2a 0xb6 0x7d 0x29 0x7d 0x2b 0xca 0x75 0x81 0x3c 0x7d 0x20 0x50 0x7d 0x24 0x7d 0x38 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x2b 0x32 0xe2 0x56 0xee 0x50 0x7d 0x30 0x7d 0x28 0x7d 0x20 0x49 0xe3 0x7d 0x20 0x7d 0x20 0x37 0x38 0x39 0x92 0xa2 0x7e

Rx_Data:_____
_____
FCS is OK!
0x7e 0x21 0x45 0x 0 0x 0 0x28 0x13 0x92 0x40 0x 0 0x30 0x 6 0xd7 0xcb 0xca 0x75 0x81 0x3c 0x a 0xb6 0x 9 0x b 0x 4 0x18 0x 0 0x50 0x32 0xe2 0x56 0xee 0x 0 0x 0 0x 0 0x e 0x50 0x10 0xff 0xf2 0xc2 0x28 0x 0 0x 0 0x2a 0x15 0x7e

Rx_Data:_____
_____
FCS is OK!
0x7e 0x21 0x45 0x 0 0x 0 0x2b 0x13 0xad 0x40 0x 0 0x30 0x 6 0xd7 0xad 0xca 0x75 0x81 0x3c 0x a 0xb6 0x 9 0x b 0x 4 0x18 0x 0 0x50 0x32 0xe2 0x56 0xee 0x 0 0x 0 0x 0 0x e 0x50 0x18 0xff 0xf2 0x57 0xe8 0x 0 0x 0 0x34 0x35 0x36 0x28 0x33 0x7e

-----TCP Checksum is Eroor!

----------Receive Data :
456
Tx_Data (IP_Packet):
Tx_Data:_____
_____
FCS is OK!
0x7e 0x21 0x45 0x7d 0x20 0x7d 0x20 0x28 0x7d 0x33 0xad 0x40 0x7d 0x20 0x30 0x7d 0x26 0xd7 0xb0 0x7d 0x2a 0xb6 0x7d 0x29 0x7d 0x2b 0xca 0x75 0x81 0x3c 0x7d 0x20 0x50 0x7d 0x24 0x7d 0x38 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x2e 0x32 0xe2 0x56 0xf1 0x50 0x7d 0x30 0x7d 0x28 0x7d 0x20 0xba 0x7d 0x38 0x7d 0x20 0x7d 0x20 0x84 0xe3 0x7e

The Data Queue is Empty

Rx_Data:_____
_____
FCS is OK!
0x7e 0x21 0x45 0x 0 0x 0 0x2b 0x13 0xcf 0x40 0x 0 0x30 0x 6 0xd7 0x8b 0xca 0x75
0x81 0x3c 0x a 0xb6 0x 9 0x b 0x 4 0x18 0x 0 0x50 0x32 0xe2 0x56 0xf1 0x 0 0x 0
0x 0 0x e 0x50 0x18 0xff 0xf2 0xeb 0xa2 0x 0 0x 0 0x71 0x77 0x65 0x9a 0x72 0x7e

-----TCP Checksum is Eroor!

----------Receive Data :
qwe
Tx_Data (IP_Packet):
Tx_Data:_____
_____
FCS is OK!
0x7e 0x21 0x45 0x7d 0x20 0x7d 0x20 0x28 0x7d 0x33 0xcf 0x40 0x7d 0x20 0x30 0x7d
0x26 0xd7 0x8e 0x7d 0x2a 0xb6 0x7d 0x29 0x7d 0x2b 0xca 0x75 0x81 0x3c 0x7d 0x20
0x50 0x7d 0x24 0x7d 0x38 0x7d 0x20 0x7d 0x20 0x7d 0x20 0x7d 0x2e 0x32 0xe2 0x56
0xf4 0x50 0x7d 0x30 0x7d 0x28 0x7d 0x20 0xba 0x7d 0x35 0x7d 0x20 0x7d 0x20 0x88
0x4e 0x7e

The Data Queue is Empty

Rx_Data:_____
_____
FCS is OK!
0x7e 0x21 0x45 0x 0 0x 0 0x28 0x13 0xe8 0x40 0x 0 0x2f 0x 6 0xd8 0x75 0xca 0x75
0x81 0x3c 0x a 0xb6 0x 9 0x b 0x 4 0x18 0x 0 0x50 0x32 0xe2 0x56 0xf4 0x 0 0x 0
0x 0 0x e 0x50 0x11 0xff 0xf2 0xc2 0x21 0x 0 0x 0 0x24 0xdb 0x7e

Tx_Data (IP_Packet):
Tx_Data:_____
_____
FCS is OK!
0x7e 0x21 0x45 0x7d 0x20 0x7d 0x20 0x28 0x7d 0x33 0xe8 0x40 0x7d 0x20 0

Contact: YunLiu

Tel: 13201517775

Institute: Computer Department, Xi’an Institute of Post and Telecommunications

Postalcode: 710061

Address: No. 563, Chang’an South Road, Xi’an, Shaanxi Province

2006-08-31