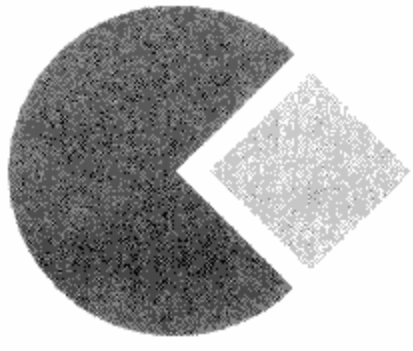


第 11 章　通用异步收发器 UART

本章目标

- 了解 UART 的原理
- 掌握 S3C2410/S3C2440 中 UART 的使用

11.1　UART 原理及 UART 部件使用方法

11.1.1　UART 原理说明

通用异步收发器简称 UART，即“Universal Asynchronous Receiver Transmitter”，它用来传输串行数据：发送数据时，CPU 将并行数据写入 UART，UART 按照一定的格式在一根电线上串行发出；接收数据时，UART 检测另一根电线上的信号，将串行收集放在缓冲区中，CPU 即可读取 UART 获得这些数据。UART 之间以全双工方式传输数据，最精简的连线方法只有 3 根电线：TxD 用于发送数据，RxD 用于接收数据，Gnd 用于给双方提供参考电平，连线如图 11.1 所示。

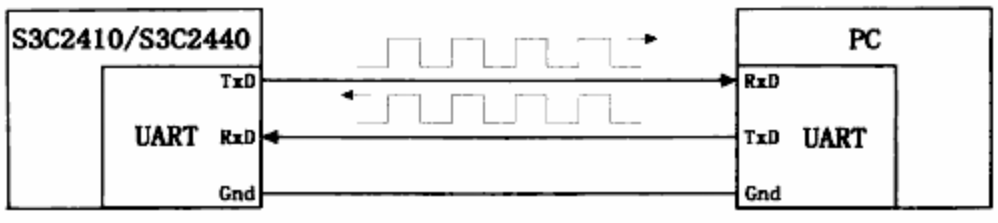


图 11.1　UART 连线图

UART 使用标准的 TTL/CMOS 逻辑电平（0～5V、0～3.3V、0～2.5V 或 0～1.8V）来表示数据，高电平表示 1，低电平表示 0。为了增强数据的抗干扰能力、提高传输长度，通常将 TTL/CMOS 逻辑电平转换为 RS-232 逻辑电平，3～12V 表示 0，−3～−12V 表示 1。

TxD、RxD 数据线以“位”为最小单位传输数据。帧（frame）由具有完整意义的、不可分割的若干位组成，它包含开始位、数据位、校验位（需要的话）和停止位。发送数据之前，UART 之间要约定好数据的传输速率（即每位所占据的时间，其倒数称为波特率）、数据的传输格式（即有多少个数据位、是否使用校验位、是奇校验还是偶校验、有多少个停止位）。

数据传输流程如下。

（1）平时数据线处于“空闭”状态（1 状态）。

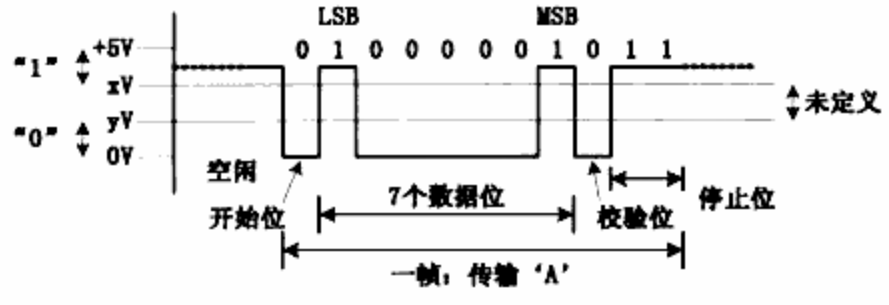
（2）当要发送数据时，UART 改变 TxD 数据线的状态（变为 0 状态）并维持 1 位的时间，这样接收方检测到开始位后，再等待 1.5 位的时间就开始一位一位地检测数据线的状态得到所传输的数据。

（3）UART 一帧中可以有 5、6、7 或 8 位的数据，发送方一位一位地改变数据线的状态将它们发送出去，首先发送最低位。

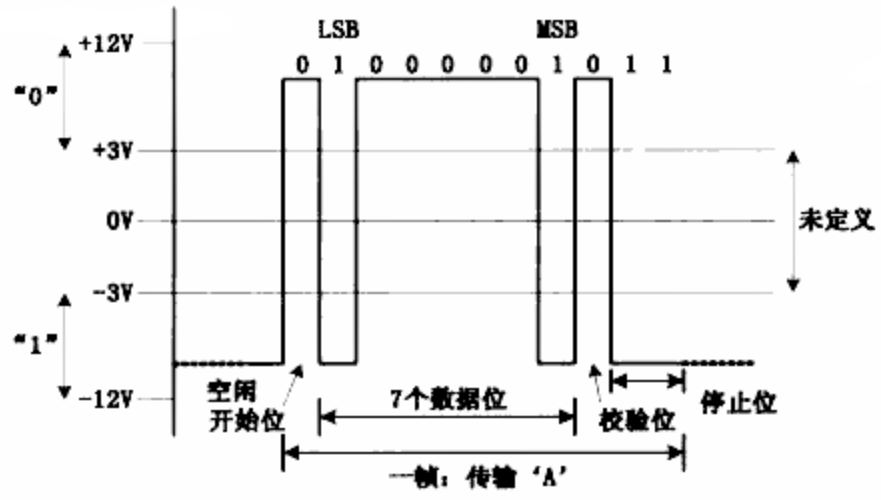
（4）如果使用校验功能，UART 在发送完数据位后，还要发送 1 个校验位。有两种校验方法：奇校验、偶校验——数据位连同校验位中，“1”的数目等于奇数或偶数。

（5）最后，发送停止位，数据线恢复到“空闭”状态（1 状态）。停止位的长度有 3 种：1 位、1.5 位、2 位。

图 11.2 演示了 UART 使用 7 个数据位、偶校验、2 个停止位的格式传输字符‘A’（二进制值为 0b1000001）时，TTL/CMOS 逻辑电平、RS-232 逻辑电平对应的波形。



1. TTL/CMOS逻辑电平下，传输‘A’时的波形



2. RS-232逻辑电平下，传输‘A’时的波形

图 11.2　UART 在 TTL/CMOS 逻辑电平和 RS-232 逻辑电平下的串行波形

11.1.2　S3C2410/S3C2440 UART 的特性

S3C2410/S3C2440 中 UART 功能相似，有 3 个独立的通道，每个通道都可以工作于中断模式或 DMA 模式，即 UART 可以发出中断或 DMA 请求以便在 UART、CPU 间传输数据。UART 由波特率发生器、发送器、接收器和控制逻辑组成。

使用系统时钟时，S3C2410 的 UART 波特率可以达到 230.4Kbit/s，S3C2440 则可以达到 115.2Kbit/s；如果使用 UEXTCLK 引脚提供的外部时钟，则可以达到更高的波特率。波特率可以通过编程进行控制。

S3C2410 UART 的每个通道都有 16 字节的发送 FIFO 和 16 字节的接收 FIFO，S3C2440

UART 的 FIFO 深度为 64。发送数据时，CPU 先将数据写入发送 FIFO 中，然后 UART 会自动将 FIFO 中的数据复制到“发送移位器”（Transmit Shifter）中，发送移位器将数据一位一位地发送到 TxDn 数据线上（根据设定的格式，插入开始位、校验位和停止位）。接收数据时，“接收移位器”（Receive Shifter）将 RxDn 数据线上的数据一位一位接收进来，然后复制到接收 FIFO 中，CPU 即可从中读取数据。

S3C2410/S3C2440 UART 的每个通道支持的停止位有 1 位、2 位，数据位有 5、6、7 或 8 位，支持校验功能，另外还有红外发送/接收功能。

S3C2410/S3C2440 UART 结构如图 11.3 所示。

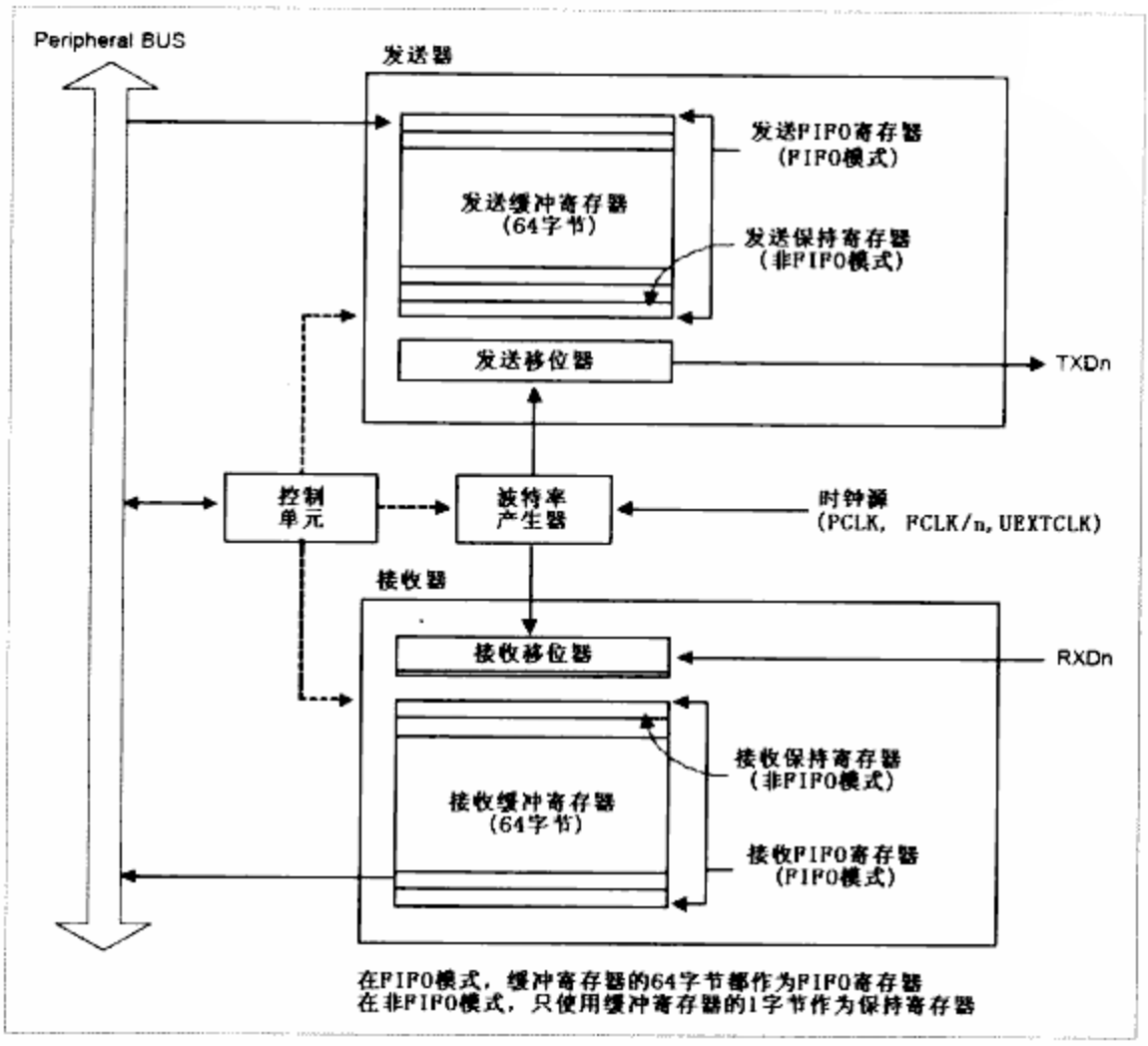


图 11.3 S3C2410/S3C2440 UART 结构图（对于 S3C2440，FIFO 深度为 64）

11.1.3 S3C2410/S3C2440 UART 的使用

在使用 UART 之前需要设置波特率、传输格式（有多少个数据位、是否使用校验位、是奇校验还是偶校验、有多少个停止位、是否使用流量控制）；对于 S3C2410/S3C2440，还要选择所涉及管脚为 UART 功能、选择 UART 通道的工作模式为中断模式或 DMA 模式。设置好之后，往某个寄存器写入数据即可发送，读取某个寄存器即可得到接收到的数据。可以通过查询状态寄存器或设置中断来获知数据是否已经发送完毕、是否已经接收到数据。

针对上述要点，下面一一说明。

1. 将所涉及的 UART 通道管脚设为 UART 功能

比如 UART 通道 0 中，GPH2、GPH3 分别用作 TXD0、RXD0，要使用 UART 通道 0 时，先设置 GPHCON 寄存器将 GPH2、GPH3 引脚的功能设为 TXD0、RXD0。

2. UBRDIVn 寄存器（UART BAUD RATE DIVISOR）：设置波特率

S3C2410 UART 的时钟源有两种选择：PCLK、UEXTCLK；S3C2440 UART 的时钟源有三种选择：PCLK、UEXTCLK、FCLK/n，其中 n 值通过 UCON0～UCON2 联合设置。

根据给定的波特率、所选择的时钟源的频率，可以通过以下公式计算 UBRDIVn 寄存器（n 为 0～2，对应 3 个 UART 通道）的值，如下所示：

```
UBRDIVn = (int)( UART clock / ( buad rate x 16) ) - 1
```

上式计算出来的 UBRDIVn 寄存器值不一定是整数，只要其误差在 1.87%之内即可。误差计算公式如下：

```
tUPCLK = (UBRDIVn + 1) x 16 x 1Frame / (UART clock)     // tUPCLK: 实际的 UART 时钟
tUEXACT = 1Frame / baud-rate                            // tUEXACT: 理论的 UART 时钟
UART error = (tUPCLK - tUEXACT) / tUEXACT x 100%        // 误差
```

3. ULCONn 寄存器（UART LINE CONTROL）：设置传输格式

ULCONn 寄存器（n 为 0～2）格式如表 11.1 所示。

表 11.1　　ULCONn 寄存器格式

功　能	位	说　明
数据位宽度	[1:0]	0b00：5 位，0b01：6 位 0b10：7 位，0b11：8 位
停止位宽度	[2]	0：一帧中有一个停止位 1：一帧中有两个停止位
校验模式	[5:3]	设置校验位的产生方法、检验方法。 0b0xx：无校验 0b100：奇校验 0b101：偶校验 0b110：发送数据时强制设为 1，接收数据时检查是否为 1 0b111：发送数据时强制设为 0，接收数据时检查是否为 0
红外模式	[6]	0：正常模式 1：红外模式

UART 通道被设为红外模式时，其串行数据的波形与正常模式稍有不同，有兴趣的读者请自行阅读数据手册。

4. UCONn 寄存器（UART CONTROL）

UCONn 寄存器用于选择 UART 时钟源、设置 UART 中断方式等。由于 S3C2410 的 UART 只有两个时钟源：PCLK 和 UEXTCLK；S3C2440 UART 有三个时钟源 PCLK、UEXTCLK、

FCLK/n，所以在时钟源的选择与设置（设置 n）方面稍有不同。

S3C2410 的 UCONn 寄存器格式如表 11.2 所示。

表 11.2　　S3C2410 的 UCONn 寄存器格式

功　能	位	说　明
接收模式	[1:0]	选择如何从 UART 接收缓冲区中读取数据。 0b00：禁止接收数据 0b01：中断方式或查询方式 0b10：DMA0 请求（只能用于 UART0） DMA3 请求（只能用于 UART2） 0b11：DMA1 请求（只能用于 UART1）
发送模式	[3:2]	选择如何将数据发送到 UART 发送缓冲区中。 0b00：禁止发送数据 0b01：中断方式或查询方式 0b10：DMA0 请求（只能用于 UART0） DMA3 请求（只能用于 UART2） 0b11：DMA1 请求（只能用于 UART1）
自环模式	[5]	自环模式就是将 TxDn 和 RxDn 在内部相连，用于自发自收。 0：正常模式，1：自环模式
接收错误状态中断使能	[6]	用于使能当发生接收错误 （比如帧错误、溢出错误） 时，产生中断。 0：出错时不产生中断 1：出错时产生中断
接收超时使能	[7]	当使用 UART FIFO 时，用于使能/禁止接收超时的中断。 0=禁止，1=使能
接收中断方式	[8]	如下情况发生时，将产生接收中断。 不使用 FIFO 时，接收到一个数据； 使用 FIFO 时，FIFO 中的数据达到 RxFIFO 的触发阈值。 中断方式如下设置。 0：脉冲 1：电平
发送中断方式	[9]	如下情况发生时，将产生发送中断。 不使用 FIFO 时，发送缓冲区变空； 使用 FIFO 时，FIFO 中的数据达到 TxFIFO 的触发阈值。 中断方式如下设置。 0：脉冲 1：电平
时钟选择	[10]	选择 UART 时钟源。 0：PCLK 1：UEXTCLK

S3C2440 的 UCONn 寄存器在 UART 时钟的选择方面与 S3C2410 有所不同，从位[10]往上的位含义不一样，并且原来的位[4]用于选择是否发出“break”信号，这些位的含义如表 11.3 所示。

表 11.3　　S3C2440 的 UCONn 寄存器中与 S3C2410 的不同之处

功　能	位	说　明
“break”信号	[4]	设置此位时，UART 会在一帧的时间里面发出一个“break”信号。 0：正常发送，1：发出“break”信号
时钟选择	[11:10]	选择 UART 时钟源。 0b00/0b10：PCLK 0b01：UEXTCLK 0b11：FCLK/n
FCLK 分频率系数	[15:12]	用来设置“FCLK/n”中的 n 值

UCON0、UCON1、UCON2 这 3 个寄存器的位[15:12]一起用来确定 n 值，它们的意义如下。

（1）UCON2[15]：“FCLK/n”使能位。

它等于 0 时，禁止使用“FCLK/n”作为 UART 时钟源；等于 1 时，可以用作 UART 时钟源。

（2）n 值的设置。

UCON0[15:12]、UCON1[15:12]、UCON2[14:12]三者用于设置 n 值，当其中一个被设成非 0 值时，其他两个必须为 0。

① n 值处于 7～21 时，UART 时钟=FCLK/（divider+6），divider 为 UCON0[15:12]的值，大于 0。

② n 值处于 22～36 时，UART 时钟=FCLK/（divider+21），divider 为 UCON1[15:12]的值，大于 0。

③ n 值处于 37～43 时，UART 时钟=FCLK/（divider+36），divider 为 UCON2[14:12]的值，大于 0。

④ UCON0[15:12]、UCON1[15:12]、UCON2[14:12]都等于 0 时，UART 时钟：FCLK/44。

5．UFCONn 寄存器（UART FIFO CONTROL）、UFSTATn 寄存器（UART FIFO STATUS）

UFCONn 寄存器用于设置是否使用 FIFO，设置各 FIFO 的触发阈值，即发送 FIFO 中有多少个数据时产生中断、接收 FIFO 中有多少个数据时产生中断。并可以通过设置 UFCONn 寄存器来复位各个 FIFO。

读取 UFSTATn 寄存器可以知道各个 FIFO 是否已经满、其中有多少个数据。

不使用 FIFO 时，可以认为 FIFO 的深度为 1，使用 FIFO 时 S3C2410 的 FIFO 深度为 16，S3C2440 的 FIFO 深度为 64。这两类寄存器各位的含义请读者查阅数据手册，后面的实例部分没有使用 FIFO。

6．UMCONn 寄存器（UART MODEM CONTROL）、UMSTATn 寄存器（UART MODEM STATUS）

这两类寄存器用于流量控制，这里不介绍。

7. UTRSTATn 寄存器（UART TX/RX STATUS）

UTRSTATn 寄存器用来表明数据是否已经发送完毕、是否已经接收到数据，格式如表 11.4 所示。缓冲区其实就是图 11.3 中的 FIFO，只不过不使用 FIFO 功能时可以认为其深度为 1。

表 11.4　　UTRSTATn 寄存器格式

功　能	位	说　明
接收缓冲区数据就绪	[0]	当接收到数据时，此位被自动设为 1
发送缓冲区空	[1]	当发送缓冲区中没有数据时，此位被自动设为 1
发送器空	[2]	当发送缓冲区中没有数据，并且最后一个数据也已经发送出去时，此位被自动设为 1

8. UERSTATn 寄存器（UART ERROR STATUS）

用来表示各种错误是否发生，位[0]～位[3]为 1 时分别表示溢出错误、校验错误、帧错误、检测到“break”信号。读取这个寄存器时，它会自动清 0。

需要注意的是，接收数据时如果使用 FIFO，则 UART 内部会使用一个“错误 FIFO”来表明接收 FIFO 中哪个数据在接收过程中发生了错误。CPU 只有在读出这个错误的数据时，才会觉察到发生了错误。要想清除“错误 FIFO”，则必须读出错误的数据，并读出 UERSTATn 寄存器。

9. UTXHn 寄存器（UART TRANSMIT BUFFER REGISTER）

CPU 将数据写入这个寄存器，UART 即会将它保存到缓冲区中，并自动发送出去。

10. URXHn 寄存器（UART RECEIVE BUFFER REGISTER）

当 UART 接收到数据时，CPU 读取这个寄存器，即可获得数据。

11.2　UART 操作实例

11.2.1　代码详解

本实例代码在/work/hardware/uart 目录下。目的是在串口上输入一个字符，单板接收到后将它的 ASCII 值加 1 后，从串口输出。

首先设置 MPLL 提高系统时钟，令 PCLK 为 50MHz，UART 将选择 PCLK 为时钟源。将代码复制到SDRAM中之后，调用main函数。这些代码与第10章相似。重点在于main函数对UART0的初始化、收发数据，这由 3 个函数来实现：uart0_init、getc 和 putc，它们在 serial.c 文件中。

1. UART 初始化

uart0_init 函数代码如下，每行都有注释，结合上面的寄存器用法即可理解。

```
07 #define PCLK            50000000    // init.c中的clock_init函数设置PCLK为50MHz
08 #define UART_CLK        PCLK        // UART0的时钟源设为PCLK
09 #define UART_BAUD_RATE  115200      // 波特率
10 #define UART_BRD        ((UART_CLK  / (UART_BAUD_RATE * 16)) - 1)
11
12 /*
13  * 初始化UART0
14  * 115200,8N1,无流控
15  */
16 void uart0_init(void)
17 {
18     GPHCON  |= 0xa0;    // GPH2、GPH3用作TXD0、RXD0
19     GPHUP   = 0x0c;     // GPH2、GPH3内部上拉
20
21     ULCON0  = 0x03;     // 波特率为115200，数据格式为:8个数据位，没有流控，1个数据位
22     UCON0   = 0x05;     // 查询方式，UART时钟源为PCLK
23     UFCON0  = 0x00;     // 不使用FIFO
24     UMCON0  = 0x00;     // 不使用流控
25     UBRDIV0 = UART_BRD; // 波特率为115200
26 }
27
```

2. 发送字符的函数

本实例不使用FIFO，发送字符前，首先判断上一个字符是否已经被发送出去。如果没有，则不断查询UTRSTAT0寄存器的位[2]，当它为1时表示已经发送完毕。于是，即可向UTXH0寄存器中写入当前要发送的字符。代码如下（宏TXD0READY被定义为“(1<<2)”）：

```
28 /*
29  * 发送一个字符
30  */
31 void putc(unsigned char c)
32 {
33     /* 等待，直到发送缓冲区中的数据已经全部发送出去 */
34     while (!(UTRSTAT0 & TXD0READY));
35
36     /* 向UTXH0寄存器中写入数据，UART即自动将它发送出去 */
37     UTXH0 = c;
```

```
38 }
39
```

3. 接收字符的函数

试图读取数据前，先查询 UTRSTAT0 寄存器的位[1]，当它为 1 时表示接收缓冲区中有数据。于是，即可读取 URXH0 得到数据。代码如下（宏 RXD0READY 被定义为“(1)”）：

```
40 /*
41  * 接收字符
42  */
43 unsigned char getc(void)
44 {
45     /* 等待，直到接收缓冲区中的有数据 */
46     while (!(UTRSTAT0 & RXD0READY));
47
48     /* 直接读取 URXH0 寄存器，即可获得接收到的数据 */
49     return URXH0;
50 }
51
```

4. 主函数

在 main 函数中，初始化完 UART0 之后，即不断地读取串口数据，并判断它是否为数字或字母。如果是的话，就将它加 1 后从串口输出。代码如下：

```
01 #include "serial.h"
02
03 int main()
04 {
05     unsigned char c;
06     uart0_init();   // 波特率 115200，8N1(8 个数据位，无校验位，1 个停止位)
07
08     while(1)
09     {
10         // 从串口接收数据后，判断其是否为数字或字母，若是则加 1 后输出
11         c = getc();
12         if (isDigit(c) || isLetter(c))
13             putc(c+1);
14     }
15
16     return 0;
17 }
```

11.2.2 实例测试

1. PC 上的串口工具推荐

Windows 下推荐使用 SecureCRT 工具，Linux 下推荐使用 kermit。

下面介绍在 Linux 下安装、使用 kermit 的方法。

确保 Linux 能连上网络，然后使用以下命令进行安装，会安装一个 kermit 命令。

```
$ sudo apt-get install ckermit
```

使用 kermit 之前，先建立一个配置文件，在/homt/book（假设用户名为 book）目录下创建名为.kermrc 的文件，内容如下：

```
set line /dev/ttyS0
set speed 115200
set carrier-watch off
set handshake none
set flow-control none
robust
set file type bin
set file name lit
set rec pack 1000
set send pack 1000
set window 5
```

然后，运行“$ sudo kermit -c”命令即可启动串口；要想关闭串口，可以先输入“Ctrl+\”，然后按住“C”键，最后输入“exit”后回车。

2. 测试方法

首先使用串口将开发板的 COM0 和 PC 的串口相连，打开 PC 上的串口工具并设置其波特率为 115200、8N1。

然后，在 uart 目录下运行 make 命令生成可执行文件 uart.bin，将它烧入 NAND Flash 后运行。

最后，在 PC 上的串口工具中输入数字或字母，可以看到输出另一个字符（加了 1）；如果输入其他字符，则无输出。

/work/hardware/stdio 目录下的程序在串口 0 上实现 printf、scanf 等函数，它使用 scanf、sscanf 和 printf 等函数从串口接收一个十进制数字序列，然后将它转换为十六进制输出。步骤与 UART 实例相似，读者可自行操作。