**《嵌入式系统》课程**

**作业二：**

**题目——中断方式实现UART收发数据**

姓名： 阮玉斌

学号： 2019112043

班级： 自动化3班

一、任务要求：

1、在串口上输入一个字符，单板接收后将他的ASCII值加1后，从串口输出。

2、UART接收、发送数据均用中断方式实现。

二、思路：

1、串口UART及其使用

实验要求使用UART实现收发数据，每个 UART 包含一个波特率发生器、发送器、接收器和一个控制单元，如图1-1所示。

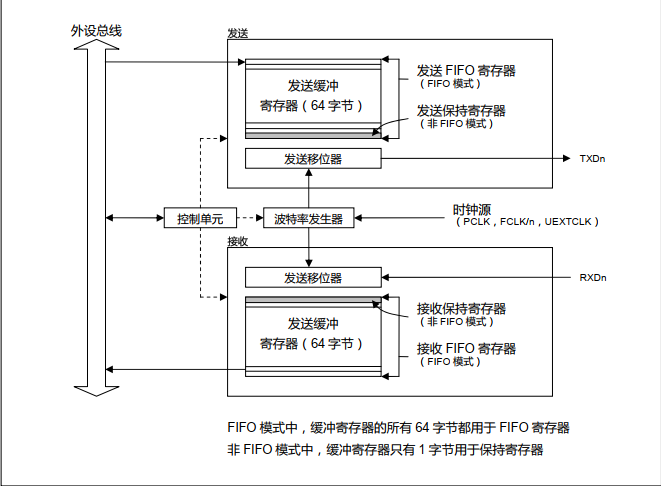


图1-1 UART方框图组成

使用串口时，需要配置主从两端。在PC端：确定串口类别，设置波特率、流控、数据位、校验位、停止位等，通过串口软件SecureCRT进行数据收发。在开发板JZ2440上，项目程序中确定开发板所用串口以及对应的不同容器，初始化串口协议配置寄存器，配置接收、发送数据函数。

2、UART中断配置

实验要求通过中断方式控制串口数据收发，本实验中结合实验要求，决定采取数据接收中断实现UART通信。对应中断源INT\_UART0(RXD0=0可服务和TXD0=1屏蔽)对应图1-2中with-sub register中断请求源。

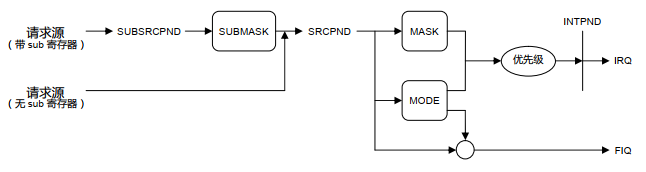
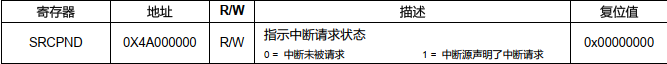


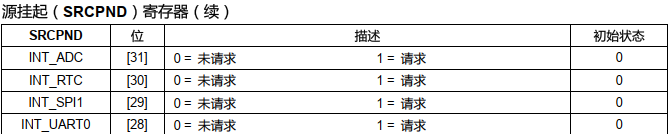
图1-2 中断处理框图

具体UART中断使用步骤如下：

（1）在“head.S”文件中设置好中断模式的栈，当发生IRQ时，CPU进入中断模式，使用中断模式下的栈；

（2）跳转执行对应INT\_UART0中断服务函数（在interrupt.c文件中）。由于本实验中用到了IRQ中断服务函数，仅采用一个中断源，通过读取INTPND和INTOFFSET寄存器确定中断源，且须在中断函数中清除中断。中断的清除过程为清除中断源的中断信号，然后清除源挂起寄存器SUBSRCPND和次级源挂起SRCPND寄存器中相应位（往相应位写1），最后清除中断挂起INTPND寄存器（往相应位写1）。本次实验采用INT\_UART0的INT\_RXD0中断，对应寄存器位如下图1-3所示，其中未列出来的其他未使用中断描述雷同，用省略号表示，本实验中该部分未使用的位均为0。

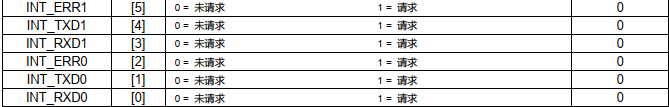


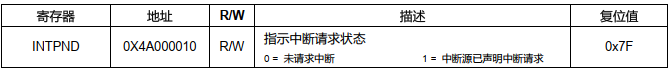


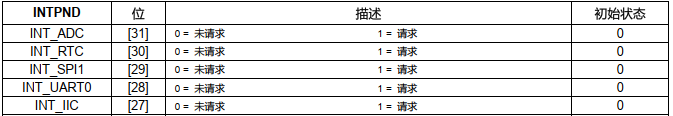
…



…







…

图1-3 INTPND、SRCPND和SUBSRCPND三个寄存器UART0中断对应位

（3）进入、退出中断模式，需要保存、回复被中断程序的运行环境。

（4）在“head.S”文件中设置CPSR寄存器中I-bit(对于IRQ）为0，使能IRQ。

（以下部分为UART0对应接收中断的初始化配置）

（5）开启UART0对应的屏蔽寄存器。

（6）确认UART0中断使用IRQ中断模式，由于本实验中仅使用一个中断，无需配置RIORITY寄存器设置优先级。

（7）设置INTSUBMSK寄存器的0位（INT\_RXD0)为0，1位（INT\_TXD0）为1，打开UART0接收中断，使能中断，同时关闭UART0发送中断，禁止中断。将INTMAK寄存器中INT\_UART0位（bit[28]）设为0，打开UART0中断屏蔽，总中断。

3、UART初始化

在使用UART之前需要配进行初始化配置，对应代码在“serial.c”文件中。具体初始化思路如下：

（1）设置波特率和数据传输格式

①通过配置线路控制ULCON0寄存器（格式如下图1-4所示）设置传输格式。

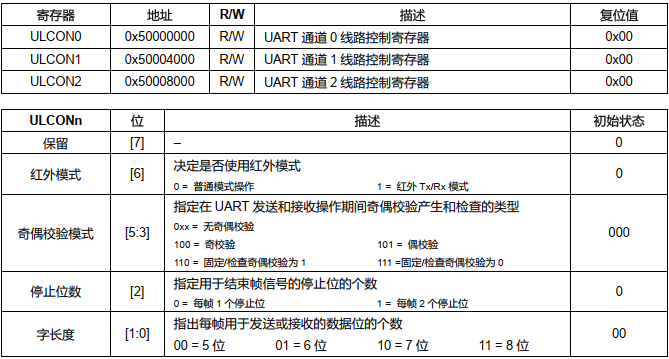
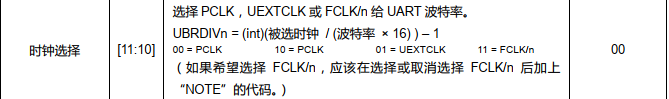


图1-4 ULCONn寄存器格式

本实验中将传输格式设置为8个数据位，无较验，1个停止位，即ULCON0=0x03。

②配置UART控制寄存器UCON0，配置UART中断方式和选择时钟源、配置波特率。UCONn寄存器格式如下图1-5所示。





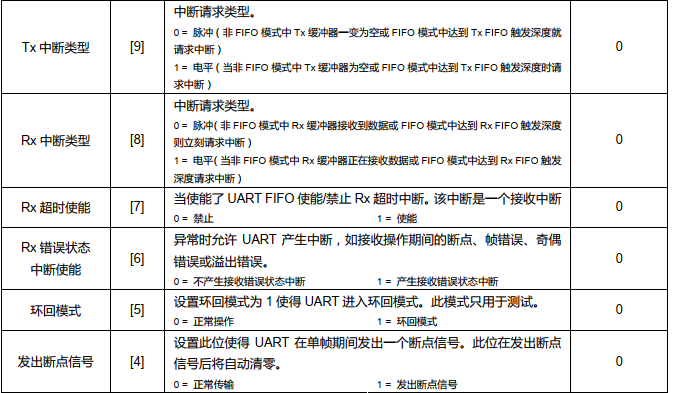


图1-5 UCONn寄存器格式

UCON0寄存器bit[12:15]的功能为FCLK分频系数，设置时钟源选择FLCK/n的分频器值。UCON0的情况：UART时钟=FCLK/(分频器+6)；分频器>0。UCON1，UCON2必须为0。本实验选择中断方式，UART时钟源为PCLK，即UCON0=0x05。

③配置UBRDIV0寄存器设置波特率为115200，给定波特率，选择时钟源频率配置计算该寄存器的值的公式如图1-5中UCONn位[11:10]的表述公式。

（2）选择涉及UART通道管脚的功能。本实验中选择UART通道0，需要先通过配置GPHCON寄存器将GPH2、GPH3引脚功能设置位TXD0,RXD0，配置GPHUP寄存器设置GPH2,GPH3为内部上拉。

（3）本实验中没有使用FIFO和流控功能，故将对应功能配置寄存器UFCON0和UMCON0均配置为0x00，不使用。

（4）清除串口串口中断挂起，清除收发中断和串口中断请求，打开UART0接收中断，关闭UART0发送中断，关闭对应中断屏蔽，使能，详细见上文中断配置部分。

4、main函数和UART0接收中断服务函数

由于实验要求使用中断进行收发，则main()中主要执行UART0串口和对应中断初始化，然后不断执行循环等待中断请求。

UART0接收中断服务函数“void UartRx\_Handle(void)”思路：

①清除串口中断请求标志；

②判断串口接收缓冲区是否数据，有数据时接收数据变量ch；

③判断ch的数据是否为数字或字母，若是则加1后输出。

三、步骤：（含代码分析、调试过程分析及验证过程图片）

1、代码分析

（1）"head.S"

@\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@ File：head.S

@ 功能：设置SDRAM，将程序复制到SDRAM，然后跳到SDRAM继续执行

@ 初始化，设置中断模式、管理模式的栈，设置好中断处理函数

@\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.extern main

.text

.global \_start

\_start:

@\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@ 中断向量，本程序中，除Reset和HandleIRQ外，其它异常都没有使用

@\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

b Reset

@ 0x04: 未定义指令中止模式的向量地址

HandleUndef:

b HandleUndef

@ 0x08: 管理模式的向量地址，通过SWI指令进入此模式

HandleSWI:

b HandleSWI

@ 0x0c: 指令预取终止导致的异常的向量地址

HandlePrefetchAbort:

b HandlePrefetchAbort

@ 0x10: 数据访问终止导致的异常的向量地址

HandleDataAbort:

b HandleDataAbort

@ 0x14: 保留

HandleNotUsed:

b HandleNotUsed

@ 0x18: 中断模式的向量地址

b HandleIRQ

@ 0x1c: 快中断模式的向量地址

HandleFIQ:

b HandleFIQ

Reset:

ldr sp, =4096 @ 设置栈指针，以下都是C函数，调用前需要设好栈

bl disable\_watch\_dog @ 关闭WATCHDOG，否则CPU会不断重启

msr cpsr\_c, #0xd2 @ 进入中断模式

ldr sp, =3072 @ 设置中断模式栈指针

msr cpsr\_c, #0xd3 @ 进入管理模式

ldr sp, =4096 @ 设置管理模式栈指针，

@ 其实复位之后，CPU就处于管理模式，

@ 前面的“ldr sp, =4096”完成同样的功能，此句可省略

bl init\_led @ 初始化LED的GPIO管脚

msr cpsr\_c, #0x53 @ 管理模式下，设置I-bit=0，开IRQ中断

bl clock\_init @ 设置MPLL，改变FCLK、HCLK、PCLK

bl memsetup @ 设置存储控制器以使用SDRAM

bl copy\_steppingstone\_to\_sdram @ 复制代码到SDRAM中

ldr pc, =on\_sdram @ 跳到SDRAM中继续执行

on\_sdram:

ldr sp, =0x34000000 @ 设置栈指针

ldr lr, =halt\_loop @ 设置返回地址

ldr pc, =main @ 调用main函数

halt\_loop:

b halt\_loop

HandleIRQ:

sub lr, lr, #4 @ 计算返回地址

stmdb sp!, { r0-r12,lr } @ 保存使用到的寄存器

@ 注意，此时的sp是中断模式的sp

@ 初始值是上面设置的3072

ldr lr, =int\_return @ 设置调用ISR即EINT\_Handle函数后的返回地址

ldr pc, =UartRx\_Handle @ 调用中断服务函数，在interrupt.c中

int\_return:

ldmia sp!, { r0-r12,pc }^ @ 中断返回, ^表示将spsr的值复制到cpsr

（2）"init.c"

/\* init.c: 对除了串口和中断外进行一些初始化 \*/

#include "s3c24xx.h"

/\* LED1对应GPF4,该灯用于区分烧录的程序是否为本次编写的程序 \*/

#define GPF4\_out (1<<(4\*2))

#define GPF4\_msk (3<<(4\*2))

void disable\_watch\_dog(void);

void clock\_init(void);

void memsetup(void);

void copy\_steppingstone\_to\_sdram(void);

void init\_led(void);

/\* 关闭WATCHDOG，否则CPU会不断重启 \*/

void disable\_watch\_dog(void)

{

WTCON = 0; // 关闭WATCHDOG很简单，往这个寄存器写0即可

}

/\* 程序烧录区分标志状态设置 \*/

void init\_led(void)

{

// LED1对应引脚设为输出

GPFCON &= ~(GPF4\_msk);

GPFCON |= GPF4\_out;

}

#define S3C2410\_MPLL\_200MHZ ((0x5c<<12)|(0x04<<4)|(0x00))

#define S3C2440\_MPLL\_200MHZ ((0x5c<<12)|(0x01<<4)|(0x02))

/\*

\* 对于MPLLCON寄存器，[19:12]为MDIV，[9:4]为PDIV，[1:0]为SDIV

\* 有如下计算公式：

\* S3C2410: MPLL(FCLK) = (m \* Fin)/(p \* 2^s)

\* S3C2440: MPLL(FCLK) = (2 \* m \* Fin)/(p \* 2^s)

\* 其中: m = MDIV + 8, p = PDIV + 2, s = SDIV

\* 对于本开发板，Fin = 12MHz

\* 设置CLKDIVN，令分频比为：FCLK:HCLK:PCLK=1:2:4，

\* FCLK=200MHz,HCLK=100MHz,PCLK=50MHz

\*/

void clock\_init(void)

{

// LOCKTIME = 0x00ffffff; // 使用默认值即可

CLKDIVN = 0x03; // FCLK:HCLK:PCLK=1:2:4, HDIVN=1,PDIVN=1

/\* 如果HDIVN非0，CPU的总线模式应该从“fast bus mode”

变为“asynchronous bus mode”\*/

\_\_asm\_\_(

"mrc p15, 0, r1, c1, c0, 0\n" /\* 读出控制寄存器 \*/

"orr r1, r1, #0xc0000000\n" /\* 设置为“asynchronous bus mode” \*/

"mcr p15, 0, r1, c1, c0, 0\n" /\* 写入控制寄存器 \*/

);

/\* 判断是S3C2410还是S3C2440 \*/

if ((GSTATUS1 == 0x32410000) || (GSTATUS1 == 0x32410002))

{

MPLLCON = S3C2410\_MPLL\_200MHZ; /\* 现在，FCLK=200MHz,HCLK=100MHz,PCLK=50MHz \*/

}

else

{

MPLLCON = S3C2440\_MPLL\_200MHZ; /\* 现在，FCLK=200MHz,HCLK=100MHz,PCLK=50MHz \*/

}

}

/\* 设置存储控制器以使用SDRAM \*/

void memsetup(void)

{

volatile unsigned long \*p = (volatile unsigned long \*)MEM\_CTL\_BASE;

/\* 这个函数之所以这样赋值，而不是像前面的实验(比如mmu实验)那样将配置值

\* 写在数组中，是因为要生成”位置无关的代码”，使得这个函数可以在被复制到

\* SDRAM之前就可以在steppingstone中运行

\*/

/\* 存储控制器13个寄存器的值 \*/

p[0] = 0x22011110; //BWSCON

p[1] = 0x00000700; //BANKCON0

p[2] = 0x00000700; //BANKCON1

p[3] = 0x00000700; //BANKCON2

p[4] = 0x00000700; //BANKCON3

p[5] = 0x00000700; //BANKCON4

p[6] = 0x00000700; //BANKCON5

p[7] = 0x00018005; //BANKCON6

p[8] = 0x00018005; //BANKCON7

/\* REFRESH,

\* HCLK=12MHz: 0x008C07A3,

\* HCLK=100MHz: 0x008C04F4

\*/

p[9] = 0x008C04F4;

p[10] = 0x000000B1; //BANKSIZE

p[11] = 0x00000030; //MRSRB6

p[12] = 0x00000030; //MRSRB7

}

void copy\_steppingstone\_to\_sdram(void)

{

unsigned int \*pdwSrc = (unsigned int \*)0;

unsigned int \*pdwDest = (unsigned int \*)0x30000000;

while (pdwSrc < (unsigned int \*)4096)

{

\*pdwDest = \*pdwSrc;

pdwDest++;

pdwSrc++;

}

}

（3）“serial.c"

#include "s3c24xx.h"

#include "serial.h"

#include "interrupt.h"

#define TXD0READY (1<<2)

#define RXD0READY (1)

#define PCLK 50000000 // init.c中的clock\_init函数设置PCLK为50MHz

#define UART\_CLK PCLK // UART0的时钟源设为PCLK

#define UART\_BAUD\_RATE 115200 // 波特率

#define UART\_BRD ((UART\_CLK / (UART\_BAUD\_RATE \* 16)) - 1)

/\*

\* 初始化UART0

\* 115200,8N1,无流控

\*/

void uart0\_init(void)

{

GPHCON |= 0xa0; // GPH2,GPH3用作TXD0,RXD0 1010,0000

GPHUP = 0x0c; // GPH2,GPH3内部上拉 1100

ULCON0 = 0x03; // 8N1(8个数据位，无较验，1个停止位) 11

UCON0 = 0x05; // 中断方式，UART时钟源为PCLK 0101

UFCON0 = 0x00; // 不使用FIFO

UMCON0 = 0x00; // 不使用流控

UBRDIV0 = UART\_BRD; // 波特率为115200

SRCPND |= 0x1<<28; //清除串口中断挂起

SUBSRCPND |= 0x3; //清除收发中断

INTPND |= 0x1<<28; //清除串口中断请求

INTSUBMSK &= ~(0x1); //打开UART0接收中断，使能中断

INTSUBMSK |= (0x1<<1); //关闭UART0发送中断，禁止中断

INTMSK &= ~(0x1<<28); //打开UART0中断屏蔽，总中断

}

/\* 发送一个字符 \*/

void putc(unsigned char c)

{

/\* 等待，直到发送缓冲区中的数据已经全部发送出去 \*/

while (!(UTRSTAT0 & TXD0READY));

/\* 向UTXH0寄存器中写入数据，UART即自动将它发送出去 \*/

UTXH0 = c;

}

/\* 接收字符 \*/

unsigned char getc(void)

{

/\* 等待，直到接收缓冲区中的有数据 \*/

while (!(UTRSTAT0 & RXD0READY)); //bit=0,无数据，值为真，循环等待

/\* 直接读取URXH0寄存器，即可获得接收到的数据 \*/

return URXH0;

}

/\* 判断一个字符是否数字 \*/

int isDigit(unsigned char c)

{

if (c >= '0' && c <= '9')

return 1;

else

return 0;

}

/\* 判断一个字符是否英文字母 \*/

int isLetter(unsigned char c)

{

if (c >= 'a' && c <= 'z')

return 1;

else if (c >= 'A' && c <= 'Z')

return 1;

else

return 0;

}

（4）"interrupt.c"

#include "s3c24xx.h"

#include "serial.h"

void UartRx\_Handle()

{

char ch;

SUBSRCPND |= 0x3; //清除rx，tx中断请求

SRCPND |= 0x1<<28; //清除串口源挂起,SRCPND第28位为INT\_UART0

//如果 SRCPND 寄存器的指定位为1，其通常被认作一个有效中断请求正在等待服务

INTPND |= 0x1<<28; //清除串口子挂起

//SUBSRCPND和SRCPND、INTPND想清除某位，均需往此位写1

if(UTRSTAT0 & 1) //判断接收缓冲区是否有数据

{

ch = URXH0; //缓冲区有数据时，接收字节数据，接受处理

//ch = getc();

//从串口接收数据后，判断其是否数字或子母，若是则加1后输出

if (isDigit(ch) || isLetter(ch))

putc(ch+1);

}

（5）"main.c"

#include "serial.h"

#include "interrupt.h"

int main()

{

uart0\_init();

// 波特率115200，8N1(8个数据位，无校验位，1个停止位)，开UART0接收中断

while(1){} //等待中断发生请求

return 0;

}

（6）"Makefile"

objs := head.o init.o serial.o interrupt.o main.o

uart.bin: $(objs)

arm-linux-ld -Tuart.lds -o uart\_elf $^

arm-linux-objcopy -O binary -S uart\_elf $@

arm-linux-objdump -D -m arm uart\_elf > uart.dis

%.o:%.c

arm-linux-gcc -Wall -O2 -c -o $@ $<

%.o:%.S

arm-linux-gcc -Wall -O2 -c -o $@ $<

clean:

rm -f uart.bin uart\_elf uart.dis \*.o

（7）涉及头文件

/\* 开发板基本固件库s3c24xx.h本报告不作记录，仅记录自定义头文件 \*/

/\* serial.h \*/

void uart0\_init(void);

void putc(unsigned char c);

unsigned char getc(void);

int isDigit(unsigned char c);

int isLetter(unsigned char c);

/\* interrupt.h \*/

void UartRx\_Handle(void);

/\* 均用作函数声明 \*/

2、调试过程分析

（1）打开FileZilla，连接VM的虚拟Ubuntu主机，将整个uart项目文件上传至Ubuntu主机路径/home/book/JZ2440/Test/uart，如下图3-1所示。



图3-1 FileZilla上传项目文件至虚拟主机

（2）Linux环境下项目编译生成“uart.bin”文件，过程如下图3-2所示。

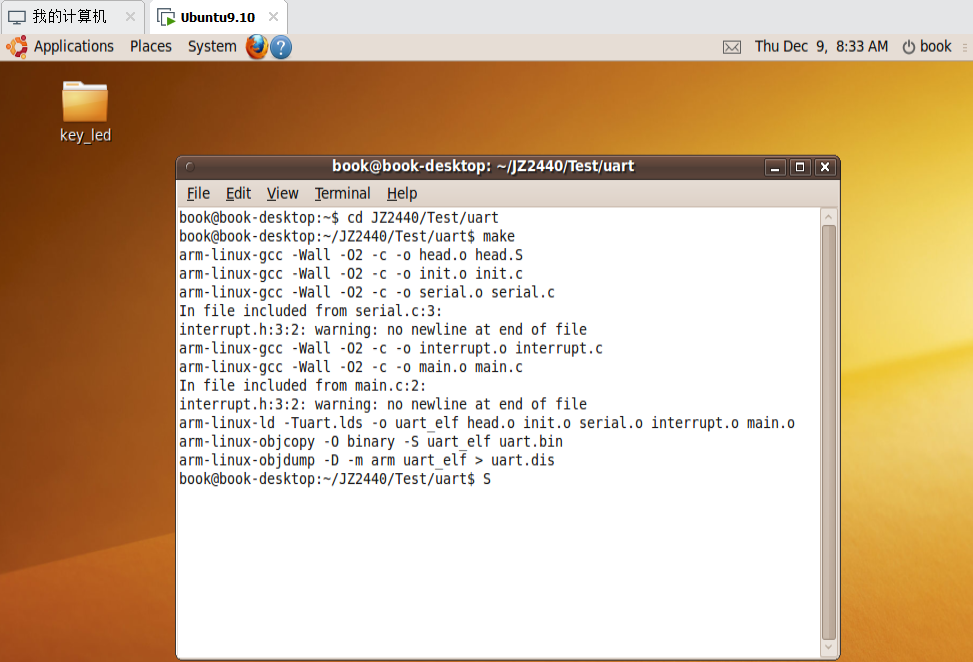


图3-2 Linux环境下编译链接项目文件

（3）将生成的“uart.bin”文件通过oflash烧录到单板。

（4）打开SecureCRT，设置对应端口波特率为115200、8N1并连接端口，按键输入数据进行串口收发功能验证。

3、验证过程与结果

通过串口调试助手，向从机端单板发送“123456789”，根据ASCII码加1对照ASCII码表，主机端可以收到返回数据“23456789：”，显示在串口通信窗口，如下图3-3所示。结果表示项目满足“用中断实现UART数据收发，且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”实验要求，通过。

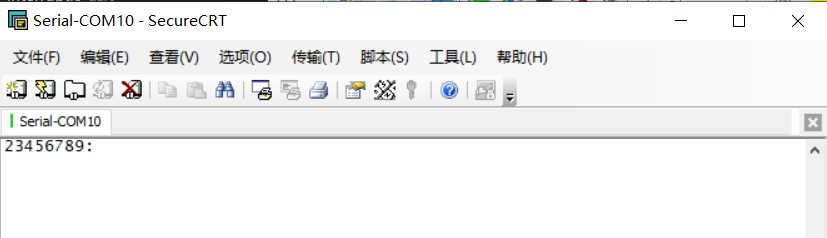


图3-3 验证过程及结果截图

四、总结：

通过本次实验，进一步掌握了嵌入式开发过程的中断控制和通用异步收发器UART的使用。UART作为协议类接口原理，其使用需通信双方均满足时序要求和共同的信号协议。UART通信和对应中断控制在使用前均需要对对应的控制寄存器进行初始化。

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。

配置中断过程：① 设置好中断模式和快速中断模式下的栈；② 准备好中断处理函数，包括异常向量表设置、中断服务程序、中断标志清除；③ 进入、退出中断或快速中断模式时需要保存、恢复中断程序运行环境；④ 对于具体中你，需要设置中断触发条件、对应引脚功能和配置对应的中断屏蔽寄存器；⑤ 确定并配置中断的使用方式；⑥ 配置CPSR寄存器中I-bit（对于IRQ）或F-bit（对于FIQ）为0，使能；⑦初始化复位中断请求挂起标志。