K9 说明书

适用范围

K9 是 ARM9-AT91RM9200 的学习板。K9 特别适用于 ARM 的初学者。

学习内容

K9 板为学习 ARM 提供了一个简单、稳定的硬件平台。
基于 K9 板，可以学习以下知识：a.硬件设计 b.Uboot 移植.c.Linux 移植 d.Linux 下的简单驱动开发。

学习环境

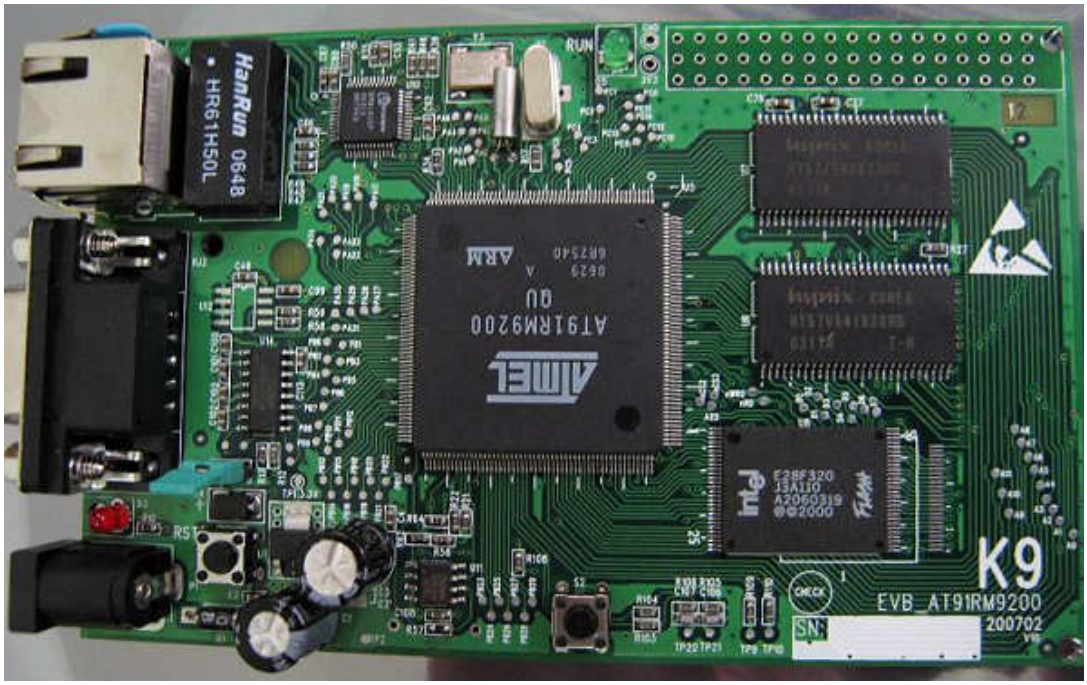
K9 板不用 JTAG 口，不需额外配备仿真器，只需要一台具有串口和网口的电脑，就可以自行搭配学习环境。在 win 下编写修改代码；在 redhat 下编译；在 win 系统自带的超级终端串口通讯工具下烧写和调试。
一切就这么简单。

购买办法

K9 学习板价格：**350.00RMB**

购买办法：上门取货或邮购

交流 QQ 群：**36825887**



接口信息

电源接口：DC5V1A
电源指示灯：红色 LED
复位按键：RST
调试串口：DB9 母座，TXD,RXD,GND 三线 RS232 电平接口。
升级跳线：跳线帽。开路，升级，调试串口打”C”；短路，正常运行。
网口：RJ45
测试按键：S2
测试指示灯：绿色 LED RUN

硬件信息

ARM9 + 4M Flash + 16M SDRAM +180MHz
CPU：AT91RM9200 封装：PQFP208 180MHz
FLASH：E28F320J3 封装：TSOP56 4M
SDRAM：HY57V641620 x 2 封装：TSOP II 54 16M
PHY：DM9161E
EEPROM：24C02
RESET：SP708S
UART：SP3232
PCB：四层

电源

输入：DC5V1A
工作电压：DC3.3V
CPU 内核电压：1.8V

文档

原理图：orcad 格式 pdf 格式
PCB 图：PowerPCB 格式
烧录文件：k9loader.bin/k9boot.bin/k9Uboot.bin/k9uImage/k9fs4m.gz
源码文件：k9loader/k9boot/k9Uboot/linux_2.4.19 内核源码
调试移植文档：

K9 实验环境

实验硬件环境一

PC + 串口线 + K9 板 + DC9V/1A 电源适配器

实验硬件环境二

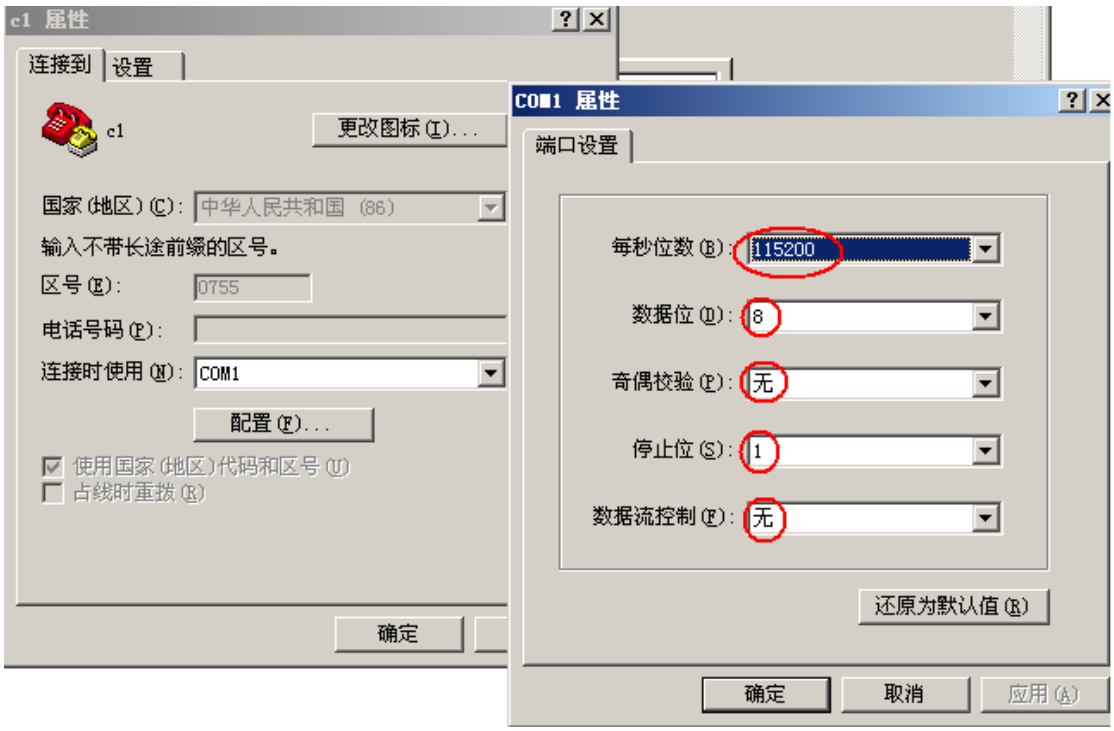
PC + 串口线 + 网线 + K9 板 + DC9V/1A 电源适配器

PC 串口和 K9 串口连接选用 2-2 ,3-3,5-5 的串口线。
PC 网口和 K9 网口直接连接时选用交叉网线。
PC 网口通过 HUB 与 K9 网口连接时选用平行网线。

实验软件环境

PC 安装以下软件：
操作系统：XP
串口调试、下载工具：超级终端
网口下载工具：TFTPSRV.EXE
编辑软件：uEdit
虚拟机软件：Vmware
编译环境：在 VM 里面安装 RedHat 9.0
请参考 **K9 调试环境**。

关于超级终端设置



K9_UBOOT 的烧写和升级

需要文件 k9loader.bin k9bboot.bin k9Uboot.bin

1. 按实验硬件环境插接好硬件。K9 板不上电。
2. 将升级短路帽至于断开位置。
3. 按超级终端设置办法打开超级终端，建立连接。
4. K9 板上电，此时在超级终端里面显示“CCCC”，即为打“C”。
5. 在超级终端下，采用 **Xmodem** 下，发送 k9loader.bin 文件，然后超级终端会出现“K9-Loader OK”的提示，然后继续出现“CCCC…”。
6. 在超级终端下，采用 **Xmodem** 下，继续发送 k9Uboot.bin 文件，发送完毕后显示 K9-UBOOT>的提示符。
7. 擦除 FLASH

   K9-UBOOT>protect off all

   K9-UBOOT>erase all
8. 装载 k9boot.bin

   K9-UBOOT >loadb 20000000 #在超级终端，用 **Kermit** 模式发送文件 k9boot.bin

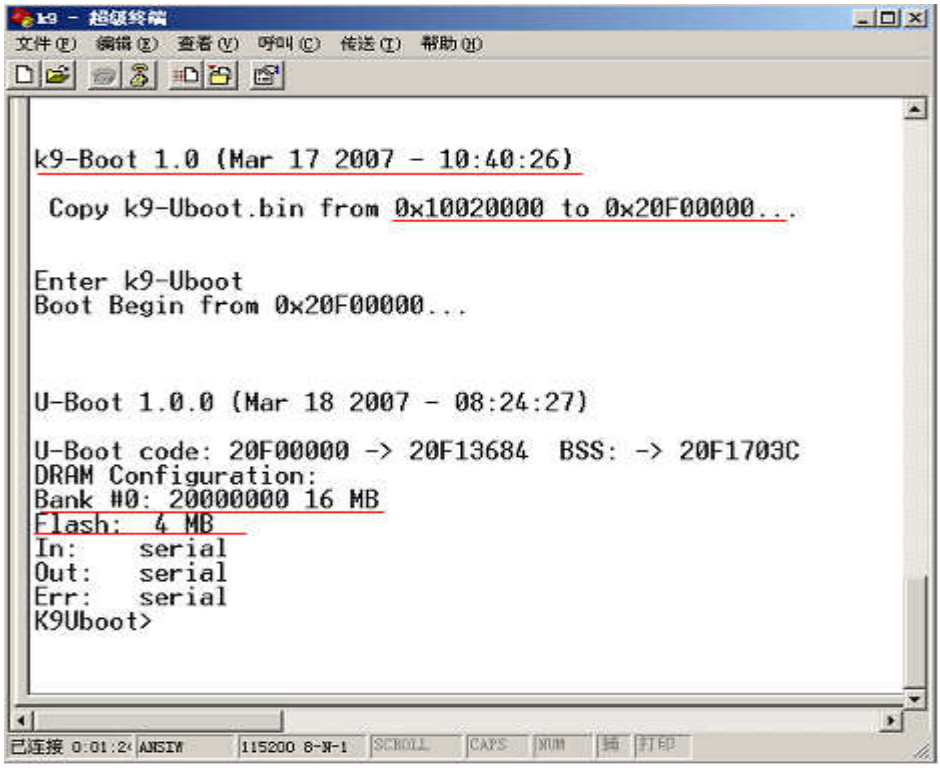
   K9-UBOOT >cp.b 20000000 10000000 5fff #在超级终端，显示拷贝的情况
9. 装载 k9Uboot.bin

   K9-UBOOT >loadb 20000000 #在超级终端，用 **Kermit** 模式发送文件 k9Uboot.bin

   K9-UBOOT >cp.b 20000000 10020000 1ffff #在超级终端，显示拷贝的情况

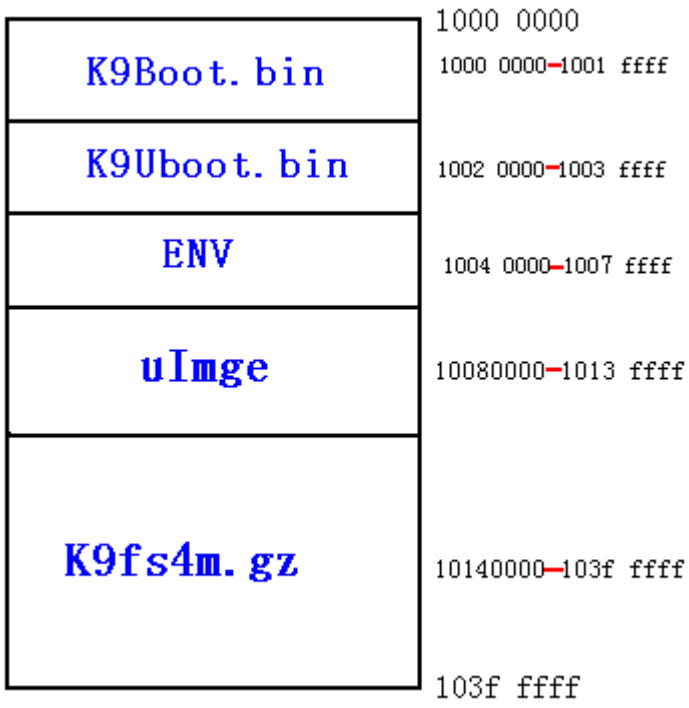
   K9-UBOOT >protect on all

   FLASH 区域保护
10. 关闭电源，将升级短路帽至于短接位置，加电复位后，超级终端接收如下：



K9 板存储空间分配方案

K9 板采用一片 E28F320 的 FLASHROM，16 位总线，共 32Mbit=4MByte 空间。32Mbit 分为 32 个扇区（sector/block）,每个扇区 128KByte(128Kbyte x 8=1024Kbit=1Mbit)。 E28F320 的地址空间为 0x1000 0000－0x103F FFFF。



K9 板采用 2 片 HY57V641620 的 SDRAM，32 位总线。共 128Mbit=16Mbyte。地址空间为 0x2000 0000－0x0x20FF FFFF。
其中，K9Uboot 运行在 20F0 0000 以后的区域。

K9 启动过程分析

1. 短路子 TS1 断开情况下
   a) CPU9200 从内置于 CPU_ROM 内部的一小段的启动代码启动，此内置启动代码主要是初始化 CPU 的内部 RAM，串口和 Xmodem 协议。之后在串口输出 CC，就是所谓的打 C。
   b) 打 C 后可以通过串口 Xmodem 传输 K9Loader.bin。此时的 k9Loader.bin 存在于 CPU_RAM 里面。K9Loader.bin 传输完成后自动运行，主要是进一步初始化 CPU，特别是 SDRAM，最后再此用 Xmodem 准备接收 Uboot。
   c) 继续传输 K9Uboot.bin，此时，Uboot 是传输存放在 SDRAM 里面。传输完成后，Uboot 在 SDRAM 里面运行起来。可以看见 K9-UBOOT >命令提示符了。
   d) K9-Uboot 启动了以后，就有了强大的功能，能够支持 flash 读写，串口下线，网口 FTP 下载，等等。
   e) 但是，到目前为止，K9Loader 是存放在 RAM 里面，K9Uboot 是存放在 SDRAM 里面，它们一掉电就会丢失。所以，就有烧写固化的问题。就是将启动代码 boot、uboot 和应用程序烧写固化到 flashrom 里面去。这部分过程请参考**烧写和升级 UBOOT** 中的第 7—9 步。
      K9Loader 体积比较小，可以存放在 CPU 内部 RAM 中运行。而 uboot 比较大，不能存到 RAM 里面，只能存到 SDRAM 里面运行。
2. 短路子 TS1 短路情况下
   a) CPU9200 从 FLASHROM 地址 1000 0000 开始运行。而 1000 0000 存放的是 K9boot。
   b) K9boot 运行起来之后先是初始化 CPU 和 SDRAM，然后将存放在 10020000 处的 K9Uboot 拷贝到 SDRAM200F0000 并运行 uboot。这个拷贝请参考 k9boot 源码 main.c 里面的## memcpy(DST,SRC,LEN); ##这段代码。
   c) 关于 K9boot 和 K9Uboot 是如何存放在 Flashrom 里面的，请参考**烧写和升级 UBOOT** 中的第 7—9 步。
      关于 K9boot、K9Uboot 存放在 FLASH 的位置，请参考 **K9 板存储空间分配方案**和 Uboot 源码 k9uboot\include\configs\at91rm9200dk.h 里面的定义。

K9 环境变量设置

K9 环境变量存放在 Flash 10040000-1007 ffff 的位置，可以通过 K9Uboot>printenv 来查看 env 的内容。当 10040000-1007ffff 为空时，uboot 启动时会显示 crc error。一般可以通过简单执行 K9Uboot>saveenv 就可以消除 crc error 的显示。

K9 环境变量设置示例：

K9Uboot>setenv ethaddr 11:22:33:44:55:66 #设置 mac 地址

K9Uboot>setenv ipaddr 192.168.1.1 #设置 K9 本机 IP 地址

K9Uboot>setenv serverip 192.168.1.2 #设置 TFTPSRV 服务器 PC 的 IP 地址

##设置以上三个环境变量后，K9 就可以通过网口进行 TFPT 下载

K9Uboot>setenv bootcmd cp.b 10140000 20a00000 2bffff\;bootm 0x10080000

cp.b 10140000 20a00000 2bfff 是将存放在 flash 10140000 位置的 ramdisk 拷贝到 SDRAM 20a00000 位置。

bootm 0x1008000 是从存放在 flash 10080000 的 uImage 处开始运行。

K9Uboot>setenv bootargs root=/dev/ram rw initrd=0x20A00000,6000000 ramdisk_size=4096 console=ttyS0,115200 mem=16M

K9Uboot>saveenv

ENV 示例：

K9Uboot> printenv
bootdelay=4
baudrate=115200
ethaddr=00:11:22:33:44:00
filesize=207299
ipaddr=192.168.1.6
serverip=192.168.1.150
bootcmd=cp.b 10140000 20a00000 2bffff;bootm 0x10080000
bootargs=root=/dev/ram rw initrd=0x20a00000,6000000 ramdisk_size=5120 console=tt
yS0,115200 mem=16M
stdin=serial
stdout=serial
stderr=serial

Environment size: 307/131068 bytes

K9_linux 的下载烧写

需要文件 k9uImage k9fs4m.gz

k9_linux 包括二个文件，内核影像文件 k9uImage 和 4M 的文件系统 k9fs4m.gz。下载 k9_linux 之前需要预先设置环境变量，详见 **K9 环境变量设置**。

在 uboot 命令提示符，可以通过串口或网口下载 k9_linux。

1． 通过串口下载 k9_linux

硬件连接串口线，启用超级终端。

a) 下载k9uImage 到SDRAM

```
K9Uboot>loadb 20000000          #kermit 协议传送k9uImage,k9uImage小于768KB
```

b) 拷贝k9uImage 到flash 10080000-1013ffff

```
K9Uboot>cp.b 20000000 10080000 cffff
```

c) 下载K9fs4m.gz 到SDRAM

```
K9Uboot>loadb 20000000          #kermit 协议传送K9fs4m.gz
```

d) 拷贝K9fs4m.gz 到flash 10140000-103fffff

```
K9Uboot>cp.b 20000000 10140000 2bffff
K9Uboot>protect on all
```

2． 通过网口下载 k9_linux

硬件连接串口线，网线，启用超级终端。

将文件 k9uImge、k9fs4m.gz、tftpsrv.exe 放置于同一个文件夹。

PC 机 IP 设置为 192.168.1.2。启动 tftpsrv.exe。

a) 下载k9uImage 到SDRAM

```
K9Uboot>tftp 20000000 k9uImge
```

b) 拷贝k9uImage 到flash 10080000-1013ffff

```
K9Uboot>cp.b 20000000 10080000 cffff
```

c) 下载K9fs4m.gz 到SDRAM

```
K9Uboot>tftp 20000000 k9fs4m.gz
```

d) 拷贝K9fs4m.gz 到flash 10140000-103fffff

```
K9Uboot>cp.b 20000000 10140000 2bffff
K9Uboot>protect on all
```

K9启动示例

```
k9-Boot 1.0 (Mar 17 2007 - 10:40:26)
  Copy k9-Uboot.bin from 0x10020000 to 0x20F00000...
Enter k9-Uboot
Boot Begin from 0x20F00000...
U-Boot 1.0.0 (Mar 17 2007 - 11:22:29)
U-Boot code: 20F00000 -> 20F13684  BSS: -> 20F1703C
DRAM Configuration:
Bank #0: 20000000 16 MB
Flash:  4 MB
In:     serial
Out:    serial
Err:    serial
Hit any key to stop autoboot:  4  3  2  1  0
## Booting image at 10080000 ...
    Image Name:
    Image Type:    ARM Linux Kernel Image (gzip compressed)
    Data Size:     547408 Bytes = 534.6 kB
    Load Address: 20008000
    Entry Point:  20008000
    Verifying Checksum ... OK
    Uncompressing Kernel Image ... OK

Starting kernel ...

Linux version 2.4.19-rmk7 (root@localhost.localdomain) (gcc version 2.95.3 20010315 (release))
#30 二 6 月 12 21:28:00 CST 2007
CPU: Arm920Tid(wb) revision 0
Machine: ATMEL AT91RM9200
On node 0 totalpages: 4096
zone(0): 4096 pages.
zone(1): 0 pages.
zone(2): 0 pages.
Kernel command line: root=/dev/ram rw initrd=0x20a00000,6000000 ramdisk_size=5120
console=ttyS0,115200 mem=16M
Calibrating delay loop... 89.70 BogoMIPS
Memory: 16MB = 16MB total
Memory: 8980KB available (1062K code, 216K data, 52K init)
Dentry cache hash table entries: 2048 (order: 2, 16384 bytes)
Inode cache hash table entries: 1024 (order: 1, 8192 bytes)
Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
Buffer-cache hash table entries: 1024 (order: 0, 4096 bytes)
Page-cache hash table entries: 4096 (order: 2, 16384 bytes)
POSIX conformance testing by UNIFIX
```

```
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
Starting kswapd
devfs: v1.12a (20020514) Richard Gooch (rgooch@atnf.csiro.au)
devfs: boot_options: 0x1
RAMDISK driver initialized: 16 RAM disks of 5120K size 1024 blocksize
PPP generic driver version 2.4.2
PPP Deflate Compression module registered
PPP BSD Compression module registered
physmap flash device: 200000 at 10000000
Using buffer write method
ttyS%d0 at MEM 0xfefff200 (irq = 1) is a AT91_SERIAL
ttyS%d1 at MEM 0xfefc4000 (irq = 7) is a AT91_SERIAL
eth0: Link now 100-FullDuplex
eth0: AT91 ethernet at 0xfefbc000 int=24 100-FullDuplex (00:aa:bb:cc:dd:00)
SmartMedia card inserted.
No NAND device found!!!
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 1024 bind 1024)
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
NetWinder Floating Point Emulator V0.95 (c) 1998-1999 Rebel.com
RAMDISK: Compressed image found at block 0
Freeing initrd memory: 5859K
VFS: Mounted root (ext2 filesystem).
Mounted devfs on /dev
Freeing init memory: 52K
route: SIOC[ADD|DEL]RT: No such process
route: SIOC[ADD|DEL]RT: Network is unreachable
# ifconfig etho 192.168.1.6
eth0: Link now 100-FullDuplex
# ping 192.168.1.150
PING 192.168.1.150 (192.168.1.150): 56 data bytes
64 bytes from 192.168.1.150: icmp_seq=0 ttl=64 time=0.8 ms
64 bytes from 192.168.1.150: icmp_seq=1 ttl=64 time=0.3 ms
64 bytes from 192.168.1.150: icmp_seq=2 ttl=64 time=0.3 ms

--- 192.168.1.150 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.3/0.4/0.8 ms
# ifconfig
eth0        Link encap:Ethernet    HWaddr 00:AA:BB:CC:DD:00
             inet addr:192.168.1.6    Bcast:192.168.1.255    Mask:255.255.255.0
```

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:4 errors:0 dropped:0 overruns:0 frame:0
TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:100
RX bytes:370 (370.0 B) TX bytes:336 (336.0 B)
Interrupt:24 Base address:0xc000