usrRoot()函数执行过程研究

SWD 聂勇

版 本 历 史

版本/状态	责任人	起止日期	备注
V1.0/正式	聂勇	2Nov2010	usrRoot()函数执行过程研究
V1.1/正式	聂勇	12Nov2010	usrRoot()函数执行过程研究

目　录

1. usrRoot()函数分析

usrRoot()函数是 vxWorks 执行的第一个任务，具有最高优先级 0，又叫做根任务。该任务中完成众多硬件的初始化。例如 flash 驱动，网络驱动等等。最后会初始化一个命令行窗口 shell。

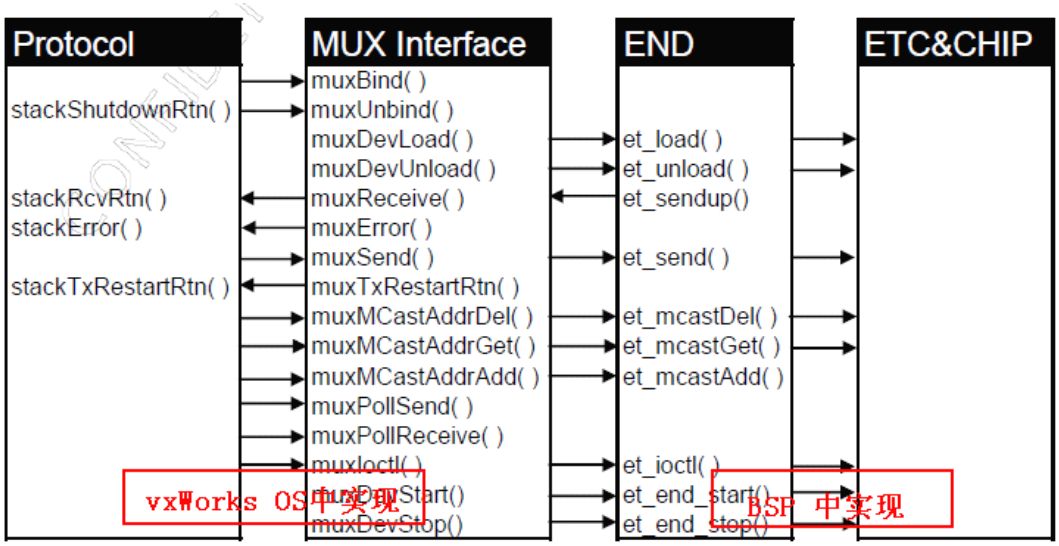
对 BSP 的简化的重点，就是这个函数。在研究这个函数的时候，有一个小小的技巧，那就是将 tmp.o 文件反汇编，然后将 usrRoot()函数提取出来，这样就可以清晰的看到什么函数被调用，不需要去看 C 代码中错杂的宏，预编译。

由于在调试过程中，遇到的问题是 bootrom 网络的问题，下面也只重点分析网络驱动这一块内容。有关其他驱动的知识，可在以后陆续补上。

2. 网络驱动

对网络驱动（ET Network driver）初始化时在最后进行的，调用函数 muxLibInit()完成。有关网络驱动部门，在硬件上涉及到两个知识点，一是 Ethernet MAC，另外一个是 DMA。作为网络驱动，主要涉及到三个模块，分别是 END 子层，ETC 子层，CHIP 子层。

整个 vxWorks 的网络堆栈和驱动（ET Network driver）如下图所示：



BSP 中与 ET Network driver 有关的 6 个源文件列表如下：

et_vx.c	END子层的实现，调用ETC子层和CHIP子层
etc.c	ETC子层的实现
etc47xx.c	CHIP子层的实现，主要是对BCM 5836的Ethernet MACs寄存器的操作
sysEtEnd.c	提供sysEtEndLoad()函数，链接et_load()函数和muxDevLoad()函数

hnddma.c	提供DMA控制器驱动
vx_osl.c	在网络驱动（ET Network driver）中使用到的工具程序

在了解了网络驱动（ET Network driver）在 BSP 中的文件之后，下面重点看 et_vx.c 文件提供的函数。这些函数也是提供给 vxWorks 中的 MUX 的基本接口函数。如下表所示：

et_vx.c 文件中主要实现的接口函数	
et_load()	Initialize the ET driver and load it into the MUX.
et_unload()	Unload the ET driver from MUX and free resources.
et_end_start()	Start the ET driver
et_end_stop()	Stop the ET driver
et_send()	Send a packet
et_ioctl()	Access the ET driver control functions
et_mcastAdd()	Add an address to the ET device's multicast address list
et_mastDel()	Delete an address from the ET device's multicast address list
et_mcastGet()	Get the list of multicast addresses maintained for the ET device

BCM5836 拥有两个 Ethernet MAC，分别为 Ethernet MAC0，Ethernet MAC1。那么，这两个模块，如何知道，哪一个是连接到交换机上的 MAC 芯片，哪一个是用来做 bootrom 时的 vxWorks 镜像的下载呢？

2.1 初始化网络驱动

网络驱动的初始化分两步进行，第一步是加载网络驱动，第二步是开启网络驱动。在 usrRoot()函数中，只有一个函数 muxLibInit()函数。无法看到这个函数的实现，和技术文档上的有些出入。

但是，整个加载的过程主要是 et_load()函数来完成。

2.1.1 et_load()加载 ET network driver

Et_load() 加载网络驱动的过程如下：

1) Malloc an et_info_t structure with END_OBJ

2) Call etc_attach()->chipattach() to initialize ETC and CHIP sub-layer

3) Call et_netpool_alloc() to set up a memory pool for receive and transmit buffers.

4) Use intConnect() to register interrupt handler.

5) Return a pointer to the initialized END_OBJ structure.

2.1.2 et_end_start()启动 ET network driver

et_end_start()启动网络驱动的过程如下：

1) Call chipreset() to reset MAC core and DMA controller.
2) Call chiptxreclaim() and chiprxreclaim() to reclaim DMA descriptions.
3) Call chipinit() to initialize MAC core and DMA controller.

2.2 DMA 控制器驱动

在这里，需要了解 DMA 的工作过程和机制，对更好的了解数据的发送接收过程有帮助。

两个 MAC Core，每一个有一个 DMA Controller。每个 DMA 控制器有两个通道，分别是 Transmit Channel 和 Receive Channel，分别负责内存数据的写出和写入。

在内存中，为每一个 DMA 控制器维持有一个 descriptor table，相关的细节可以了解 DMA 的驱动。

3. bootCmdLoop

在任务 usrRoot 的最后，创建了一个优先级为 1 的任务，这个任务的目的就是实现 bootrom 的命令行窗口，使用户可以通过命令行来控制 vxWorks 的启动。

bootConfig.c

```
……
    taskSpawn ("tBoot", bootCmdTaskPriority, bootCmdTaskOptions,
        bootCmdTaskStackSize, (FUNCPTR) bootCmdLoop,
        0,0,0,0,0,0,0,0,0,0);
……
```

可以看到，该任务函数为 bootCmdLoop()，该函数也是在 bootCofig.c 文件中实现。

bootCmdLoop()函数的实现过程

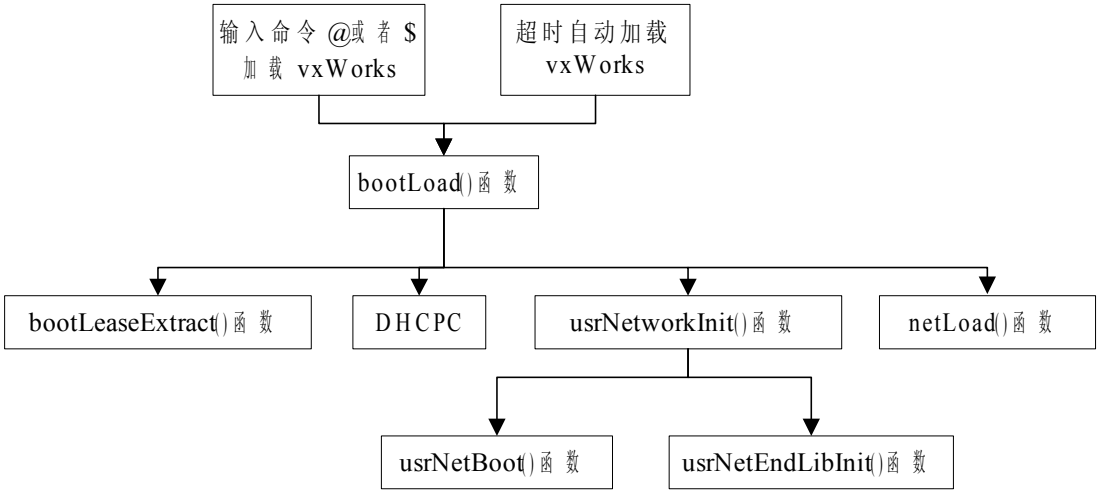
1	usrBootLineInit()	根据启动类型（COLD　WARM），调用函数 sysNvRamGet()函数进行初始化。
2	printExcMsg()	打印异常信息，在命令行中输入 e 命令，也是调用该函数打印出异常信息。
3	bootStringToStruct()	开始启动自动加载 vxWorks 镜像，除非被中断。该函数的实现应该是在 vxWorks 中。在 BSP 中无法找到实现。
4	autoboot()	查询时间是否到了，是否可以开始自动加载 vxWorks 镜像。

5	switch　case	下面就是命令行界面，接收用户输入的命令行，完成相关的操作。

那么，bootrom 是如何加载 vxWorks 的呢？或者说，在命令行上配置好网络参数，输入命令@之后，bootrom 是如何将 vxWorks 的镜像通过网络下载到 ram 中，然后将控制权交给 vxWorks 镜像？下面就根据这个线索，来了解 vxWorks 镜像的下载过程。

3.1 vxWorks 镜像的下载

系统自动加载和在命令行输入@加载的函数调用关系大致相同。下面首先给出其函数调用关系图。



上图只给出了几个主要的函数，其中 usrNetworkInit()函数用于网络的配置，例如 et_load()驱动函数的注册等等。在最后，会调用 netLoad()函数来发送完成代码的 ftp 下载，也可以这么说，是 netLoad()函数中有发送和接收数据包的操作。

参考《BSP 简化及调试研究.docx》，加载时出现异常问题就出在 usrNetworkInit()函数中。而发送不出数据包的问题，就出在 netLoad()函数。