1. 概述

嵌入式系统由硬件环境、嵌入式操作系统和应用程序组成，硬件环境是操作系统和应用程序运行的硬件平台，它随应用的不同而有不同的要求。硬件平台的多样性是嵌入式系统的主要特点，如何使嵌入式操作系统在不同的硬件平台上有效地运行，是嵌入式系统开发中需要解决的关键问题。解决的方法是在硬件平台和操作系统之间提供硬件相关层来屏蔽这些硬件的差异，给操作系统提供统一的运行环境，这种硬件相关层就是嵌入式系统中的板级支持包BSP(Board Support Package，简称BSP)。

2. BSP及其作用

BSP是嵌入式系统中介于硬件平台和操作系统之间的中间层软件，主要目的是为了屏蔽底层硬件的多样性，根据操作系统的要求完成对硬件的直接操作，向操作系统提供底层硬件信息并最终启动操作系统。BSP具有硬件相关性和操作系统相关性的特点，其主要作用包括：

> 1. 初始化底层硬件，为操作系统提供底层硬件信息；
> 2. 初始化相关硬件设备，主要是存储设备、通信设备；
> 3. 检测系统硬件是否正常；
> 4. 加载操作系统并启动系统运行。

3. 嵌入式Linux系统BSP的实现

BSP是相对于操作系统而言的，不同的操作系统有不同定义形式的BSP，要求BSP所实现的功能也有所不同。在嵌入式Linux系统中，主要是初始化底层硬件并引导操作系统；同时，BSP又是和硬件相关的，还要考虑对硬件的初始化操作。
在不同的开发阶段，因为核心和文件系统所处的位置不同，BSP所要完成的工作也有所不同： 在开发调试阶段，BSP要能够与主机通信并从主机下载核心；
在目标产品中，BSP要能够从非易失存储设备中加载核心。

3.1 开发调试阶段BSP的实现

开发初期由于调试系统的需要，BSP需把核心和文件系统从主机直接下载到目标板的内存中运行. BSP要完成如下工作： **1. 硬件的初始化和配置 2. 通信设备的初始化** BSP需与主机通信，从主机下载核心和文件系统，因此要完成相应通信设备的初始化。与主机通信的设备一般是网卡和串口。
串口间通信要遵循一定的协议，包括数据格式、同步方式、传输速率、纠错方式等。对串口的初始化就是对这些协议进行设定，使通信双方处于相同的传输模式。在目标板上初始化串口是通过设置其寄存器实现的：

- 设置串口的行控制寄存器确定串口接收数据的格式;
- 设置串口的波特率产生寄存器确定串口接收数据的速率;
- 设置串口的通信协议（八个数据位、一个停止位，没有奇偶校验位，9600波特率）。

串口初始化后就可以从其数据接收寄存器中读取数据。
对网卡初始化也是通过设置其寄存器实现的，设置控制寄存器，使网卡处于接收模式。用网卡与主机通信时，主机端的通信程序要知道目标板上网卡的MAC地址才能发送数据。因此，我们要把网卡的MAC地址设定为指定值。
从网卡的数据接收寄存器读取数据时要把数据包中的非数据信息(包的状态、长度、原地址、目的地址和类型)

丢掉。 BSP从主机接收文件，因此必须要提供主机与BSP通信的程序。主机端的通信程序可用操作系统提供的系统调用直接设置串口的属性，使主机端串口的通信协议与目标板串口的通信协议一致。主机端的程序通过与目标板连接的串口线将数据写到目标板串口的数据寄存器中。用网卡时，用原始套接口对网卡进行写操作，把数据包发送到目标板上网卡的数据接收缓存寄存器中。 **3. 从主机接收核心和文件系统，启动核心运行** 系统加电时，BSP从位于0地址的非易失存储器FLASH中执行，和运行在主机上的程序通信，从串口或网卡的数据寄存器中读取数据，把核心和文件系统下载到内存中指定的位置，最后将CPU中的程序计数器PC置为核心在内存中的起始地址，实现核心启动。但是，程序在FLASH中执行时不能对变量进行写操作，为了使程序能正确执行，BSP必须将自己重定位(即把自己搬运到)到内存中，并且在进入c语言函数执行前要设置好堆栈指针。 其主要实现过程的伪码如下：

```
硬件(cpu、内存等)初始化;
通信设备(网卡、串口)初始化;
将自己重定位到内存中;
设置系统的堆栈指针;
跳转到从串口读取核心和文件系统的函数;
从串口读取核心和文件系统的函数(void)
{
    while(核心没读取完){
        while(串口的接收数据寄存器为空)(等待);
        从串口的数据寄存器读取数据到内存中;
        核心大小减去已读取的大小，确定核心是否读取完;
        if(核心读取完){
            -asm{mov pc，一核心在内存中的起始地址；}
    }
}
```

3.2. 目标产品中BSP的实现

3.2.1 BSP独立实现

把核心和文件系统直接从主机下载到内存中运行，只适用于开发调试阶段。目标产品中核心和文件系统烧写在非易失性存储设备上，因此BSP要从这些设备中加载并启动核心。此时，BSP不需要与主机通信，可以将其单独实现在产品中。将BSP单独实现时，可以根据需要向其中灵活地添加多种功能：启动核心前检测内存是否能被正确读写，通过判断网卡、声卡等硬件的属性寄存器确定硬件设备是否正常等。
此时，BSP要完成的工作如下：

> 1. 初始化硬件及存储设备。
> 2. 测试硬件设备是否正常。
> 3. 从相应的存储设备中加载核心到内存中，并启动核心.

硬件的初始化和配置与前面相同，主要完成CPU和内存的初始化。此时，BSP要从存储设备中加载并启动核心，因此要对存储设备(一般是FLASH或CF卡)进行初始化，使其能被正确寻址。BSP中读取核心的代码与具体的操作系统及文件格式无关，不能从文件系统层把核心作为一个文件读进来，只能从硬件接口来实现具体的操作，把核心从存储设备读入内存，然后把核心的开头当作一段程序的起点，使CPU转入核心执行.
非易失性存储器FLASH和内存统一寻址，对它的访问和访问内存是一样的，可以利用寄存器将核心直接从FALSH读取到内存。CF卡相对内存来说属于外设，对其进行读取操作是通过控制其寄存器(数据寄存器、状态与控制寄存器)来实现的，向其控制寄存器发布ATA命令OA何处读取，读取数据的大小等)后，判断其状态寄存器

是否准备好，才能从其数据寄存器中读取数据到内存中。把核心从非易失性存储设备读到内存中后，将CPU中的程序计数器PC置为核心在内存中的起始地址，实现系统的启动。
对应的伪码如下：

```
硬件(cpu、内存等)初始化;
存储设备(FLASH、CF卡)初始化;
测试硬件设备是否正常;
根据CF卡的属性寄存器判断CF卡是否存在;
如果存在，跳转到从CF卡加载核心到内存的函数;
如果不存在，直接将核心从FLASH加载到内存中;
mov pc，一核心在内存中的起始地址;
从CF卡加载核心到内存的函数(void)
{
    while(核心没加载完){
        向CF卡发布ATA命令，确定从CF卡的哪一块开始读，
        读多少块
        while(CF卡的状态没准备好){等待; }
        从CF卡的数据寄存器读数据到内存中;
        核心大小减去所读的块数乘以块大小;
        if(核心加载完){
            -asm{mow pc, =核心在内存中的起始地址; }
    }
}
```

3.2.2 在核心中实现BSP

BSP单独实现易于修改，当硬件改变时，只需要对相关硬件的初始化代码进行修改并重新编译。但是，对于嵌入式系统的存储介质FLASH，有些文件系统对它的分区有字节对齐性的要求，也就是说分区必须是特定大小或特定大小的倍数才能有写访问权限，如果把BSP单独写在一个分区中会造成存储空间的浪费。实质上，BSP是属于操作系统的一部分且和操作系统绑定在一起运行，在开发板硬件固定的情况下，可以将其实现在Linux核心中。在核心中实现BSP时，BSP应能向核心提供必要的底层硬件信息，实现核心从非易失性存储器FLASH到内存的加载和启动。
在存储介质FLASH中没有所谓的引导扇区，相应的嵌入式Linux内核映象zImage也没有如PC机上的引导扇区代码bootsect及辅助代码setup，而是由head.o、misc.o、head-xscale.o和piggy.o几个文件顺序连接而成。其中，head.o是由 / arch / arm / boot / compressed / head.s汇编而成，是核心最先执行的代码，主要作用是用misc.o解压缩核心并使CPU转到核心解压缩后所在的内存地址执行。head-xscale.o是体系结构相关的代码。piggy.o是系统启动后保留在内存中的全部有用程序[3]，也就是head.o要解压缩的代码。由以上分析可知，嵌入式Linux的内核不能自启动，要启动核必须满足以下两个条件：

> 1. 系统硬件已被正确初始化;
> 2. 核心程序都在内存中，并且CPU中的程序计数器被置为核心在内存中的起始地址。

在核心中实现BSP时，BSP必须位于核心代码的最前面，即将其实现在head.S文件的最前面，完成如PC 机上的BIOS、bootsect和setup的功能。同时，要保证核心在加载到内存后必须能跳转到和没有BSP时核心中相同的地方执行。重新编译核心后，就将BSP实现在核心映象zImage中的开始代码中。把核心烧到FLASH的0地址，系统启动时首先执行上述代码，将核心加载到内存中执行，实现核心的自启动。