



# 飞腾嵌入式 LINUX 用户 手册

(V1.5.2)

2022 年 03 月

飞腾信息技术有限公司

[www.phytium.com.cn](http://www.phytium.com.cn)

## 版权声明：

本文档用于指导用户的相关应用和开发工作，版权归属飞腾信息技术有限公司所有，受法律保护。任何未经书面许可的公开、复制、转载、篡改行为将被依法追究法律责任。

## 免责声明：

本文档仅提供阶段性数据，并不保证该等数据的准确性及完整性。飞腾信息技术有限公司对此文档内容享有最终解释权，且保留随时更新、补充和修订的权利。所有资料如有更改，恕不另行通知。

如有技术问题，可联系 [support@phytium.com.cn](mailto:support@phytium.com.cn) 获取支持，因不当使用本文档造成的损失，本公司概不承担任何责任。

## 当前版本

文件标识	
当前版本	<b>1.5.2</b>
完成日期	2022.03

## 版本历史

版本	修订时间	修订人	修订内容
V1.0	202006		
V1.1	202008		
V1.2	202011		
V1.3	202101		
V1.4	202110		
V1.5	202112	魏珊珊	使用新的文档模板，修改目录结构，更新 Yocto、Mesa、Qt 版本，添加 vpu 功能，添加已知缺陷。
V1.5.1	202201	魏珊珊	增加 Xenomai 和 Xen 系统镜像，增加 Linaro 编译内核的步骤，修改 Uboot 启动时 initrd 的大小。
V1.5.2	202203	魏珊珊	在常见问题中增加“编译可扩展 SDK”的步骤；修改 UEFI 启动时拷贝内核的命令

## 目录

1	概述.....	5
2	软硬件特性.....	6
3	安装盘使用.....	9
4	开发环境配置及系统构建.....	11
4.1	硬件配置 .....	11
4.2	开发软件安装 .....	11
4.2.1	repo 发布 .....	11
4.2.2	ISO 发布.....	12
4.3	系统镜像构建 .....	13
5	系统镜像运行.....	15
5.1	Uboot 启动 .....	15
5.1.1	ramdisk 启动 .....	15
5.1.2	U 盘或 SATA (NVME) 盘启动.....	17
5.2	UEFI 启动 .....	19
6	其它系统镜像的构建和运行.....	24
6.1	Xenomai .....	24
6.1.1	系统构建 .....	24
6.1.2	运行 .....	25
6.2	Xen .....	25
6.2.1	系统构建 .....	25
6.2.2	运行 .....	25
7	系统镜像定制.....	28
7.1	内核定制 .....	28
7.2	文件系统定制 .....	28
7.3	BSP 定制 .....	30
7.4	软件包管理 .....	30
8	常用软件.....	32
8.1	Docker 使用 .....	32
8.2	实时内核测试 .....	33
8.3	图形和显式 .....	34
8.4	多媒体 .....	34
9	附录.....	36

9.1	常见问题 .....	36
9.2	编译环境配置 .....	37
9.2.1	基于 Yocto 生成交叉编译工具链 .....	37
9.2.2	下载 Linaro 的交叉编译工具链 .....	37
9.3	已知缺陷 .....	38
9.4	参考 .....	38
9.5	支持外设列表 .....	38

Phytium 飞腾

## 图目录

图 2.1	FT-2000/4 参考板功能框图.....	8
图 5.1	分区命令.....	19
图 5.2	删除分区.....	20
图 5.3	分区格式.....	20
图 5.4	第一个分区.....	20
图 5.5	第二个分区.....	21
图 5.6	分区结果.....	21
图 5.7	格式化分区.....	22
图 7.1	hello 文件夹目录结构.....	28
图 7.2	查找 core-image-minimal.bb.....	29
图 7.3	编辑 core-image-minimal.bb.....	29
图 8.1	搜索容器.....	32
图 8.2	拉取容器.....	32
图 8.3	运行容器.....	33
图 8.4	查看当前系统运行过的容器.....	33
图 8.5	删除指定容器.....	33
图 8.6	查看当前系统的容器镜像.....	33
图 8.7	删除当前系统指定镜像.....	33
图 8.8	cyclictest 的测试结果 .....	34

## 表目录

表 4-1	发布的 ISO 文件.....	12
表 4-2	可选镜像 .....	13
表 5-1	各开发板对应的内核及文件系统 .....	17
表 9-1	FT2000/4 和 FT2000/2 的 Phytium Embedded Linux V1.5.....	38
表 9-2	D2000 的 Phytium Embedded Linux V1.5 .....	39

## 1 概述

嵌入式 Linux 是将日益流行的 Linux 操作系统进行裁剪修改，使之能在嵌入式计算机系统上运行的一种操作系统。嵌入式 Linux 既继承了 Internet 上无限的开放源代码资源，又具有嵌入式操作系统的特性。

嵌入式 Linux 的特点是版权费免费，全世界的自由软件开发提供技术支持，网络特性免费而且性能优异，软件移植容易，代码开放，有许多应用软件支持，应用产品开发周期短，新产品上市迅速，因为有许多公开的代码可以参考和移植。

Yocto Project™ 是一个开源的协作软件，提供模板、工具和方法帮你创建定制的 Linux 系统和嵌入式产品，而无需关心硬件体系。适合嵌入式 Linux 开发人员使用。

本文档主要介绍通过 Yocto 工程如何构建针对飞腾 FT2000 及 D2000 系列板卡的嵌入式 Linux 内核镜像及文件系统。用户可以使用 Yocto 项目的组件进行开发，构建，调试和测试完整的嵌入式 Linux 系统。

## 2 软硬件特性

D2000 Embedded Linux 1.5 的主要功能如下：

- 工作于 ARM v8 AARCH64 模式
- 基于 Linux4.19.115
- 支持 Yocto3.3
- 支持 Mesa 版本 21.0.3
- 支持 UEFI 和 Uboot 启动
- 支持 Qt5.15.2
- 支持 Qt OpenGL（基于 X11）
- 支持 Yocto 桌面
- 支持 OpenGL3.3
- 支持 OpenGL ES3.1
- 支持 Docker
- 支持 Linux 实时补丁
- 支持板载 UART
- 支持板载以太网
- 支持 X100 显卡
- 支持 PCIE 扩展

FT-2000/4 Embedded Linux 1.5 的主要功能如下：

- 工作于 ARM v8 AARCH64 模式
- 基于 Linux4.19.115
- 支持 Yocto3.3
- 支持 Mesa 版本 21.0.3
- 支持 Uboot 和 UEFI 启动
- 支持 Qt5.15.2
- 支持 Qt OpenGL（基于 X11）
- 支持 Yocto 桌面
- 支持 OpenGL3.3
- 支持 OpenGL ES3.1
- 支持 Docker
- 支持 Linux 实时补丁
- 支持板载 UART
- 支持 SATA 硬盘



- 支持 NVME 硬盘
- 支持板载以太网
- 支持 USB 鼠标键盘
- 支持 Radeon R600 系列 PCIE 显卡
- 支持 PCIE 扩展 USB3.0
- 支持 PCIE 扩展 SATA
- 支持 RTC
- 支持板载声卡

FT-2000/2 Embedded Linux 1.5 的主要功能如下：

- 工作于 ARM v8 AARCH64 模式
- 基于 Linux4.19.115
- 支持 Yocto3.3
- 支持 Mesa 版本 21.0.3
- 支持 Uboot 启动
- 支持 Qt5.15.2
- 支持 Qt OpenGL (基于 X11)
- 支持 Yocto 桌面
- 支持 OpenGL3.3
- 支持 OpenGL ES3.1
- 支持 Docker
- 支持 Linux 实时补丁
- 支持板载 UART
- 支持板载以太网
- 支持 Radeon R600 系列 PCIE 显卡
- 支持 PCIE 扩展 USB3.0
- 支持 PCIE 扩展 SATA

参考版功能框图如下所示：

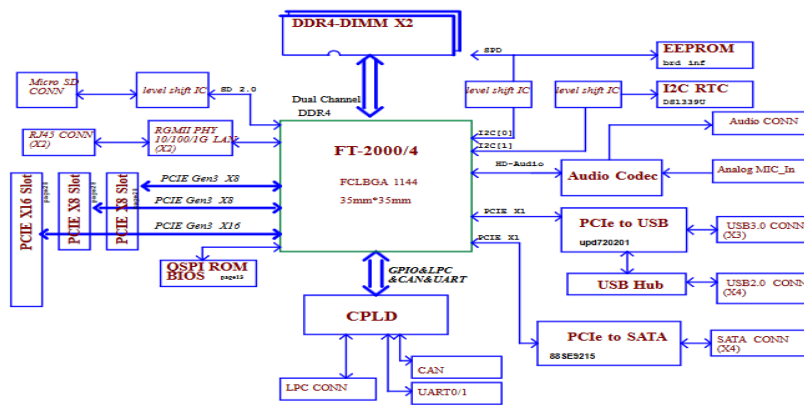


图 2.1 FT-2000/4 参考板功能框图

### 3 安装盘使用

飞腾提供定制的基于 X11 的桌面系统，通过本章可把飞腾定制的桌面系统安装到硬盘中，供用户快速测试使用。（目前只支持 FT2000/4,D2000 开发板。如果想要运行自行编译的内核和文件系统，可跳过此节。）

在 ISO 里找到 core-image-anaconda-<machine>.iso 文件，把文件写入 USB 直接启动，也可通过编译 core-image-anaconda 获得 core-image-anaconda-<machine>.iso，ISO 文件具体使用方法如下：

1、在主机端插入 USB 盘，运行下列命令制作启动盘（请按照插入 U 盘识别的设备名来更改，下列命令以/dev/sdb 为例）：

（1）使用 UEFI 启动：

```
$ sudo dd if=core-image-anaconda-<machine>.iso of=/dev/sdb bs=1M
```

（2）使用 Uboot 启动，Uboot 不能识别 ISO，需要下列步骤，烧写 USB：

```
$ sudo mount -o loop core-image-anaconda-<machine>.iso media/
```

```
$ sudo mkfs.ext4 -L boot /dev/sdb1
```

```
$ sudo mount /dev/sdb1 /mnt
```

```
$ sudo rsync -a -P media/ /mnt/
```

```
$ sudo umount /mnt
```

2、启动盘插入开发板的 USB 接口，启动：

（1）UEFI 启动模式下，BIOS 里选择 USB 设备启动。

文件系统启动成功后，显示图型安装界面，选择默认安装方式，安装成功后，重启开发板。UEFI 自动启动安装成功的文件系统。

（2）Uboot 启动模式下，在串口控制台设置 Uboot 环境变量：

```
=>mw.l 0x28180200 0x22884444 （此命令只在 FT2000/4 设置）
```

```
=>setenv bootargs root=/dev/ram0 rootdelay=5 ro initrd=0x93000000,90M
```

```
=>usb start;ext4load usb 0:1 0x90100000 <machine>.dtb;ext4load usb 0:1 0x90200000 Image;ext4load usb 0:1 0x93000000 initrd;booti 0x90200000 - 0x90100000 （对于 D2000 开发板，此命令中的<machine>.dtb 为 ft2000.dtb）
```

文件系统启动成功后，显示图型安装界面，选择默认安装方式，安装过程提示写 bootloader 错误，请选 Yes 选项。安装成功后，重启开发板，串口控制台设置 Uboot 下 SATA 盘或固态硬盘启动变量：

```
=>mw.l 0x28180200 0x22884444 （此命令只在 FT2000/4 设置）
```

```
=>setenv bootargs root=/dev/mapper/openembedded_<machine>-root rootdelay=5 rw initrd=0x93000000,90M console=ttyAMA1,115200
```

```
=>ext4load scsi (或 nvme) 0:1 0x90100000 <machine>-devboard-dsk.dtb;ext
```

```
4load scsi (或 nvme) 0:1 0x90200000 Image;ext4load scsi (或 nvme) 0:1 0x930  
00000 initrd; booti 0x90200000 - 0x90100000
```

Phytium飞腾

## 4 开发环境配置及系统构建

### 4.1 硬件配置

- 最少 4GB 内存
- Ubuntu 系统（建议使用 18.04 LTS）
- 磁盘剩余空间至少 50 GB

### 4.2 开发软件安装

首先 Ubuntu 系统中需要安装如下软件包：

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib \
build-essential chrpath socat cpio python3 python3-pip python3-pexpect \
xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa \
libssl1.2-dev pylint3 xterm libncursesw5-dev openssl libssl-dev
```

飞腾嵌入式 Linux 发行版以 repo 和 ISO 两种形式发布。

#### 4.2.1 repo 发布

##### 1、安装 repo 工具

```
$ mkdir ~/bin
$ curl https://mirrors.tuna.tsinghua.edu.cn/git/git-repo > ~/bin/repo
$ export REPO_URL='https://mirrors.tuna.tsinghua.edu.cn/git/git-repo'
$ chmod a+x ~/bin/repo
$ export PATH=${PATH}:~/bin
```

##### 2、下载 Yocto 层

```
$ export PATH=${PATH}:~/bin
$ mkdir <release-path>
$ cd <release-path>
$ git config --global user.name "Your Name"
$ git config --global user.email "email@example.com"
$ sudo ln -s /usr/bin/python3 /usr/bin/python
$ repo init -u https://gitee.com/phytium_embedded/phytium-sdk.git -b master
$ repo sync --force-sync
$ . ./setup-env -m <machine> ( Supported machines: d2000 ft2002-evm
```

ft2002hke-evm ft2004-evm)

##### 3、下载并解压 linux-4.19.tar.gz 到 path-to-your-source-tree

添加下列变量到 local.conf

```
INHERIT += "externalsrc"
```

```
EXTERNALSRC_pn-linux-phytium = "path-to-your-source-tree"
```

注意： 获得 linux-phytium 内核源码请联系飞腾嵌入式软件部。

#### 4.2.2 ISO 发布

包含以下三个文件：

表 4-1 发布的 ISO 文件

phytium-sdk-<version>-source-<yyyymmdd>-yocto.iso	源码包，包含可编译的源码，客户可从源码来构建整个系统
phytium-sdk-<version>-cache-<yyyymmdd>-yocto.iso	Cache 安装包，包含预编译的二进制文件，可加快源码构建时的编译速度
phytium-sdk-<version>-image-<yyyymmdd>-yocto.iso	二进制安装包，包含编译好的 image, dtb, rootfs

1、从源码编译构建系统需要安装以下的 ISO：

```
$ sudo mount -o loop phytium-sdk-<version>-source-<yyyymmdd>-yocto.iso /mnt
$ cd /mnt
$ ./install
```

2、选择安装 cache ISO 以加快编译速度：

```
$ sudo mount -o loop phytium-sdk-<version>-cache-<yyyymmdd>-yocto.iso /mnt
$ cd /mnt
$ ./install
```

3、按照以下步骤安装 image ISO 获取编译好的系统文件：

```
$ sudo mount -o loop phytium-sdk-<version>-image-<yyyymmdd>-yocto.iso /image
$ cd /image
```

### 4.3 系统镜像构建

如果使用 ISO 发布中的二进制安装包 `phytium-sdk-<version>-image-<yyyymmdd>-yocto.iso`，获取编译好的内核及文件系统，请跳过此节。

可构建的镜像为：

表 4-2 可选镜像

名字	说明	提供层
core-image-minimal	最小文件系统	poky
core-image-weston	Wayland + weston	poky
core-image-xfce	Xfce 轻量级桌面环境	meta-phytium

构建系统镜像：

（注意： 请不要用 root 用户执行下列操作）

#### 1、设置开发板环境变量

```
$ cd phytium-sdk-<version>-<yyyymmdd>-yocto or phytium-sdk
```

```
$ ./setup-env -m <machine> （设置 FT2000/4/D2000 开发板环境变量）
```

#### 2、修改配置文件

- 在不能连接 Internet 的环境下需要使用安装 ISO，同时需要设置 `conf / local.conf` 文件中如下变量：

```
BB_NO_NETWORK = "1"
```

- 如果编译带 Linux 实时补丁（RT）的 image 需要在 `conf/local.conf` 文件中添加下列变量：

```
PREFERRED_PROVIDER_virtual/kernel = "linux-phytium-rt"
```

- `core-image-xfce` 和 `core-image-weston` 文件系统，默认不支持 X100 gpu 和 vpu。

如果开发板支持 X100 gpu，修改 `meta-phytium/meta-bsp/conf/machine/phytium-base.inc` 文件，添加 gpu 功能重新编译文件系统：

```
MACHINE_FEATURES ?= "pci ext2 ext3 serial usbhost alsa touchscreen  
keyboard gpu"
```

如果开发板支持 vpu，

（1）修改 `meta-phytium/meta-bsp/conf/machine/phytium-base.inc` 文件，添加 vpu 功能：

```
MACHINE_FEATURES ?= "pci ext2 ext3 serial usbhost alsa touchscreen  
keyboard vpu"
```

（2）修改 `meta-phytium/meta-bsp/conf/layer.conf` 文件，去掉：

```
BBMASK += "meta-bsp/recipes-graphics/vpu/*"
```

- 编译 weston 的文件系统, 需要在 conf/local.conf 中设置下列变量:

```
DISTRO_FEATURES_remove = " x11"
```

### 3、编译镜像

```
$ bitbake -c menuconfig virtual/kernel (进入 kernel 选项配置)
```

```
$ bitbake core-image-minimal (编译内核和文件系统, 可选文件系统见表表 4-2)
```

编译完成后用户可以在 tmp/deploy/images/<machine>/下找到编译好的镜像。

Phytium 飞腾



## 5 系统镜像运行

可以通过 Uboot 或者 UEFI 启动系统镜像。

### 5.1 Uboot 启动

#### 5.1.1 ramdisk 启动

ramdisk 只可启动最小文件系统，如需启动其它文件系统，请查看下节。

##### 1、前提条件：

- 主机串口线连接板卡的串口
- 主机和板卡连接网线
- 主机安装串口调试工具（Kermit、putty、minicom 等）
- 主机搭建 TFTP Server

其中，主机搭建 TFTP Server 的步骤如下：

（1）在开发环境 host 侧（Ubuntu）安装 tftp 服务

```
$ sudo apt-get install tftp-hpa tftpd-hpa
```

```
$ sudo apt-get install xinetd
```

（2）tftp 服务器目录为编译的镜像目录，如 /home/username/repo-release/build\_d2000/tmp/deploy/images/d2000，确保目录有执行权限 `chmod 777 /home/username/repo-release/build_d2000/tmp/deploy/images/d2000`

（3）配置主机 tftpboot 服务，新建并配置文件/etc/xinetd.d/tftp

```
$ sudo vim /etc/xinetd.d/tftp
```

```
# /etc/xinetd.d/tftp
```

```
server tftp
{
    socket_type = dgram
    protocol = udp
    wait = yes
    user = root
    server = /usr/sbin/in.tftpd
    server_args = -s /home/username/repo-release/build_d2000/tmp/deploy/images/d2000
    disable = no
    per_source = 11
    cps = 100 2
```

```
flags = IPv4
```

```
}
```

(4) 启动主机 tftp 服务，生成默认配置

```
$ sudo service tftpd-hpa start
```

(5) 修改主机 tftp 配置，修改/etc/default/tftpd-hpa

```
$ sudo vim /etc/default/tftpd-hpa
```

```
# /etc/default/tftpd-hpa
```

```
TFTP_USERNAME="tftp"
```

```
TFTP_DIRECTORY="/home/username/repo-  
release/build_d2000/tmp/deploy/images/d2000"
```

```
TFTP_ADDRESS=":69"
```

```
TFTP_OPTIONS="-l -c -s"
```

(6) 重启主机 tftp 服务

```
$ sudo service tftpd-hpa restart
```

(7) 测试主机 tftp 服务的可用性

登录 tftp 服务，获取 d2000 目录下的一个文件

```
$ tftp 127.0.0.1
```

```
tftp> get <d2000 目录下的一个文件>
```

```
tftp> q
```

## 2、用 ramdisk 启动 core-image-minimal 文件系统

(1) 主机配置 IP，重启 tftp 服务

```
$ sudo ifconfig enp2s0 <tftp_serverip>
```

```
$ sudo service tftpd-hpa restart
```

(2) 启动电源，串口输出 Uboot 命令行，设置 Uboot 环境变量（将主机 IP 和开发板 IP 配置在一个网段）

```
=>setenv ipaddr <board_ipaddr>
```

```
=>setenv serverip <tftp_serverip>
```

针对 FT2000/4:

```
=>mw.l 0x28180200 0x22884444
```

```
=>setenv bootargs console=ttyAMA1,115200 earlycon=pl011,0x28001000 ro  
ot=/dev/ram0 rw ramdisk_size=0x2000000
```

针对 D2000:

```
=>setenv bootargs console=ttyAMA1,115200 earlycon=pl011,0x28001000 root=/dev/ram0 rw ramdisk_size=0x2000000
```

针对 FT2000/2:

```
=>setenv bootargs console=ttyS0,115200 earlycon=uart8250, mmio32,0x70001000 root=/dev/ram0 rw ramdisk_size=0x2000000
```

(3) 将文件系统下载到 FT2000/4 开发板内存（如为 FT2000/2 或 D2000 开发板，请根据下表替换相应的文件）

```
=>tftpboot 0x90100000 ft2004-devboard-d4-dsk.dtb ;
```

```
=>tftpboot 0x90200000 Image
```

```
=>tftpboot 0x93000000 core-image-minimal-ft2004-evm.ext2.gz.u-boot
```

表 5-1 各开发板对应的内核及文件系统

FT2000/4	D2000	FT2000/2
Image	Image	uImage
ft2004-devboard-d4-dsk.dtb	d2000-devboard-dsk.dtb	ft2000ahk-devboard-dsk.dtb
core-image-minimal-ft2004-evm.ext2.gz.u-boot	core-image-minimal-d2000.ext2.gz.u-boot	core-image-minimal-ft2002-evm.ext2.gz.u-boot

(4) FT2000/4 和 D2000 使用 booti 命令启动，FT2000/2 使用 bootm 命令启动，请根据板卡型号选择输入下列命令：

```
=>booti 0x90200000 0x93000000 0x90100000 (FT2000/4, D2000)
```

或:

```
=>bootm 0x90200000 0x93000000 0x90100000 (FT2000/2)
```

### 5.1.2 U 盘或 SATA (NVME) 盘启动

由于 FT2000/2 板卡没有板载 USB 及硬盘，以下步骤仅针对 FT2000/4、D2000 开发板。使用 U 盘或 SATA (NVME) 盘，可启动表 4-2 中的所有文件系统，仍以 core-image-minimal 文件系统为例。

#### 1、前提条件:

- 主机串口线连接板卡的串口
- 主机安装串口调试工具（Kermit、putty、minicom 等）

#### 2、在主机端分区和格式化 USB/SATA 设备:

以主机识别设备名为/dev/sdb 为例，请按实际识别设备名更改，nvme 设备请改为 nvme 的设备名，或参考下一节（UEFI 启动）中分区和格式化步骤。

分区:

```
$ sudo fdisk /dev/sdb
```

输入下列参数，每个参数都跟回车

```
p          [打印分区表]
d          [删除分区]
n          [添加新分区]
p          [主分区]
1          [第一个分区]
2048       [分区起始扇区]
+1G        [第一个分区大小]
p          [检查分区]
n          [添加新分区]
p          [主分区]
2          [第二个分区]
2200000    [分区起始扇区]
+100G      [第二个分区大小]
p          [打印分区表]
w          [将分区表写入磁盘并退出]
```

格式化设备，并将文件系统拷贝到设备：

```
$ sudo mkfs.ext4 /dev/sdb1
```

```
$ sudo mkfs.ext4 /dev/sdb2
```

```
$ sudo mount /dev/sdb1 /mnt
```

```
$ cd <image-path> (<image-path>为 tmp/deploy/images/<machine>)
```

```
$ sudo cp Image ft2004-devboard-d4-dsk.dtb /mnt (D2000 为 d2000-devboard-
dsk.dtb)
```

```
$ sudo cp initrd.img.rootfs.cpio.gz /mnt/initrd.img
```

```
$ sudo umount /dev/sdb1
```

```
$ sudo mount /dev/sdb2 /mnt
```

```
$ sudo cp core-image-minimal.tar.gz /mnt
```

```
$ cd /mnt
```

```
$ sudo tar zxvf core-image-minimal.tar.gz
```

```
$ cd ~
```

```
$ sudo umount /dev/sdb2
```

### 3、设置 Uboot 环境变量：

- 文件系统在 U 盘上（以 U 盘启动后识别为/dev/sda 为例，请按实际识别

设备名更改 root= 参数):

```
=>setenv bootargs console=ttyAMA1,115200 earlycon=pl011,0x28001000
root=/dev/sda2 rootdelay=5 rw initrd=0x93000000,90M
```

```
=>usb start
```

```
=>ext4load usb 0:1 0x90100000 ft2004-devboard-d4-dsk.dtb
```

```
=>ext4load usb 0:1 0x90200000 Image
```

```
=>ext4load usb 0:1 0x93000000 initrd.img
```

```
=>booti 0x90200000 - 0x90100000
```

● 文件系统在 SATA 硬盘上 (以 SATA 盘启动后识别为/dev/sda 为例, 请按实际识别设备名更改 root= 参数, 如果是在 nvme 盘上启动, 把 ext4load scsi 改为 nvme, root=/dev/nvme0n1p2):

```
=>setenv bootargs console=ttyAMA1,115200 earlycon=pl011,0x28001000
root=/dev/sda2 rootdelay=5 rw initrd=0x93000000,90M
```

```
=>ext4load scsi 0:1 0x90100000 ft2004-devboard-d4-dsk.dtb
```

```
=>ext4load scsi 0:1 0x90200000 Image
```

```
=>ext4load scsi 0:1 0x93000000 initrd.img
```

```
=>booti 0x90200000 - 0x90100000
```

## 5.2 UEFI 启动

目前基于 FT2000/4、D2000 的开发板, 有 Uboot 和 UEFI 两种引导方式, 本节将说明 UEFI 的启动方法。FT2000/2 的开发板只有 Uboot 启动, 请跳过此节。

### 1、前提条件

- 一台基于 Linux 发行版的主机。
- 一块硬盘 (固态硬盘或者机械硬盘均可) 或 U 盘均可 (将制作成 UEFI 引导盘, 会格式化硬盘, 注意保存原有数据)。

### 2、制作步骤

(1) 在终端下找到对应硬盘盘符, 在本文档中, 该硬盘为/dev/sdb, 注意根据实际情况替换相应设备。

(2) 在 root 用户下使用 fdisk 分区, 如下所示。

```
root@sxf-OptiPlex-7070:/home/sxf# fdisk /dev/sdb
```

图 5.1 分区命令

(3) 使用 p 命令查看当前分区, 若存在分区, 使用 d 命令删除。

```
Command (m for help): d
Partition number (1,2, default 2): 1

Partition 1 has been deleted.

Command (m for help): d
Selected partition 2
Partition 2 has been deleted.

Command (m for help):
```

图 5.2 删除分区

(4) 确认删除所有分区后，输入 `g` 命令标记硬盘分区格式为 GPT（UEFI 分区格式）。

```
Command (m for help): g
Created a new GPT disklabel (GUID: 76710B1B-695C-CF49-96EA-5FADFDD8953A).
The old dos signature will be removed by a write command.

Command (m for help):
```

图 5.3 分区格式

(5) 之后使用 `n` 命令开始分区，第一个分区大小为 500M，为 EFI 分区，输入命令如下，首先是分区号，默认为 1，回车即可；之后是起始扇区，回车即可；之后是尾部扇区，输入 `+500M`，表明分区大小为 500M，扇区地址自动计算。之后使用 `t` 命令设置分区类型，输入 1 表明是 EFI 分区，具体操作如下图所示。

```
Command (m for help): n
Partition number (1-128, default 1):
First sector (2048-1953525134, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-1953525134, default 1953525134): +500M

Created a new partition 1 of type 'Linux filesystem' and of size 500 MiB.

Command (m for help): t
Selected partition 1
Partition type (type L to list all types): 1
Changed type of partition 'Linux filesystem' to 'EFI System'.

Command (m for help):
```

图 5.4 第一个分区

(6) 完成第一个分区后，在使用 `n` 命令创建第二个分区，操作类似，分区大小自行设定，操作如下所示。

```
Command (m for help): n
Partition number (2-128, default 2):
First sector (1026048-1953525134, default 1026048):
Last sector, +sectors or +size{K,M,G,T,P} (1026048-1953525134, default 1953525134): +20G

Created a new partition 2 of type 'Linux filesystem' and of size 20 GiB.

Command (m for help):
```

图 5.5 第二个分区

(7) 其他分区操作类似，分区完成后使用 `p` 命令查看当前分区情况，如下所示。

```
Command (m for help): p
Disk /dev/sdb: 931.5 GiB, 1000204886016 bytes, 1953525168 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disklabel type: gpt
Disk identifier: 76710B1B-695C-CF49-96EA-5FADFDD8953A

Device            Start      End          Sectors      Size Type
/dev/sdb1          2048      1026047      1024000      500M EFI System
/dev/sdb2         1026048    42969087    41943040      20G Linux filesystem
/dev/sdb3         42969088    462399487    419430400    200G Linux filesystem
/dev/sdb4         462399488    881829887    419430400    200G Linux filesystem
/dev/sdb5         881829888   1953525134  1071695247    511G Linux filesystem

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.

root@sxf-0ptiPlex-7070:/home/sxf#
```

图 5.6 分区结果

(8) 确认没有问题后，使用 `w` 命令保存设置并退出，若需要删除分区，可以使用 `d` 命令，按照提示操作即可。

(9) 分区完成后，使用 `mkfs.vfat` 格式化 EFI 分区，使用 `mkfs.ext4` 格式化其他分区，如下图所示。



```

root@sx-f-OptiPlex-7070:/home/sxf# mkfs.vfat /dev/sdb1
mkfs.fat 4.1 (2017-01-24)
root@sx-f-OptiPlex-7070:/home/sxf# mkfs.ext4 /dev/sdb2
mke2fs 1.44.1 (24-Mar-2018)
Creating filesystem with 5242880 4k blocks and 1310720 inodes
Filesystem UUID: 72e94b6b-7015-4783-ad07-675086b266ae
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
    4096000

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

root@sx-f-OptiPlex-7070:/home/sxf# █

```

图 5.7 格式化分区

(10) 挂载根文件系统分区，操作与上述类似，将根文件系统包拷贝到对应分区即可，飞腾提供示例根文件系统包，用户也可选用自己编译的镜像，只需将根文件系统解压到硬盘第二个分区即可，示例如下：

- 挂载分区：`sudo mount /dev/sdb2 /mnt2`
- 挂载 EFI 分区：`sudo mount /dev/sdb1 /mnt`
- 切换到镜像目录：`cd <image-path>`（<image-path>为 `tmp/deploy/images/<machine>`）
- 拷贝文件：`sudo cp core-image-minimal.tar.gz /mnt2`
- 解压文件：`sudo tar zxvf core-image-minimal.tar.gz -C /mnt2`
- 拷贝内核：`sudo cp -rL /mnt2/boot/* /mnt`  
`sudo cp Image /mnt`
- 修改引导文件：`gedit /mnt/EFI/BOOT/grub.cfg`，示例文件如下所示：
 

```

serial --unit=0 --speed=115200 --word=8 --parity=no --stop=1

search --no-floppy --set=root -l 'boot'

default=boot

timeout=10

menuentry 'boot'{
  linux /Image LABEL=boot root=/dev/sda2
  initrd /initrd

```

主要将 `root` 参数修改为对应的根文件系统分区，如在本例中第二个分区为根文件系统，则将 `root` 参数修改为 `root=/dev/sda2`（默认只有一个硬盘，如果有多个硬盘，建议使用对应分区的 UUID）。

黑色加粗部分为修改的内容，可根据需要仿照上述格式添加多个表项。



- 同步: `sync`
- 卸载分区: `sudo umount /mnt /mnt2`

(11) 引导盘制作好后, 在 UEFI 下将该硬盘设为默认启动盘即可运行系统。

Phytium 飞腾

## 6 其它系统镜像的构建和运行

本章介绍 Xenomai 和 Xen 系统镜像的构建和运行。

### 6.1 Xenomai

#### 6.1.1 系统构建

##### 1、编译 Xenomai cobalt 模式

(1) 首先根据 4.2 节获得 phytium-sdk

(2) 设置开发板环境变量

```
$ cd phytium-sdk-<version>-<yyyymmdd>-yocto or phytium-sdk
```

```
$ . setup-env -m <machine> （设置 FT2000/4/D2000 开发板环境变量）
```

(3) 修改配置文件

修改 conf/local.conf 文件，添加：

```
PREFERRED_PROVIDER_virtual/kernel = "linux-xenomai-phytium"
```

(4) 编译 core-image-rt 文件系统

```
$ bitbake core-image-rt
```

编译完成后用户可以在 tmp/deploy/images/<machine>/下找到编译好的镜像。

##### 2、编译 Xenomai mercury Preempt-RT 模式

(1) 首先根据 4.2 节获得 phytium-sdk

(2) 设置开发板环境变量

```
$ cd phytium-sdk-<version>-<yyyymmdd>-yocto or phytium-sdk
```

```
$ . setup-env -m <machine> （设置 FT2000/4/D2000 开发板环境变量）
```

(3) 修改配置文件

①修改 conf/local.conf 文件，添加：

```
PREFERRED_PROVIDER_virtual/kernel = "linux-phytium-rt"
```

②修改 sources/meta-phytium/meta-xenomai/recipes-xenomai/xenomai/xenomai

\_3.1.2.bb，将

```
EXTRA_OECONF += " --enable-pshared --enable-smp \
                --with-core=cobalt"
```

修改为

```
EXTRA_OECONF += " --enable-pshared --enable-smp \
                --with-core=mercury"
```

(4) 编译 core-image-rt 文件系统

```
$ bitbake core-image-rt
```

编译完成后用户可以在 `tmp/deploy/images/<machine>/` 下找到编译好的镜像。

### 6.1.2 运行

同 5.1.2，只是将 `core-image-minimal.tar.gz` 替换为 `core-image-rt-ft2004-evm.tar.gz` 或者 `core-image-rt-d2000.tar.gz`。

## 6.2 Xen

### 6.2.1 系统构建

1、首先根据 4.2 节获得 `phytium-sdk`

2、设置开发板环境变量

```
$ cd phytium-sdk-<version>-<yyyymmdd>-yocto or phytium-sdk
```

`$ . setup-env -m <machine>` （目前只支持 FT2000/4 开发板，`<machine>` 设置为 `ft2004-evm`）

3、修改配置文件

修改 `conf/local.conf` 文件，添加：

```
DISTRO_FEATURES_append = " xen"
```

4、编译 Xen 文件系统

```
$ bitbake xen-image-minimal
```

编译完成后用户可以在 `tmp/deploy/images/<machine>/` 下找到编译好的镜像。

### 6.2.2 运行

#### 1、启动 Xen 和 Dom0

（1）根据 5.1.2 节，在主机端将 SATA（或 NVME）盘分区，创建 2 个分区。

（2）格式化分区，将编译好的 `Image`、`xen.dtb`、`xen-ft2004-evm.efi` 拷贝到第一个分区，将文件系统 `xen-image-minimal-ft2004-evm.tar.gz` 拷贝到第二个分区。

```
$ sudo mkfs.ext4 /dev/sdb1
```

```
$ sudo mkfs.ext4 /dev/sdb2
```

```
$ sudo mount /dev/sdb1 /mnt
```

```
$ cd <image-path> (<image-path>为 tmp/deploy/images/<machine>)
```

```
$ sudo cp Image xen.dtb xen-ft2004-evm.efi /mnt
```

```
$ sudo umount /dev/sdb1
```

```
$ sudo mount /dev/sdb2 /mnt
```

```
$ sudo cp xen-image-minimal-ft2004-evm.tar.gz /mnt
```

```
$ cd /mnt
```

```
$ sudo tar zxvf xen-image-minimal-ft2004-evm.tar.gz
```

```
$ cd ~
```

```
$ sudo umount /dev/sdb2
```

(3) 在 Uboot 中依次执行如下命令启动 Xen 和 Dom0, /chosen/module@0 下的 reg 字段指定了 dom0 Linux 内核的内存位置和大小, 需要根据实际加载来调整。以 SATA 盘启动为例。

```
=>mw.l 0x28180200 0x22884444;
```

```
=>ext4load scsi 0:1 0x90200000 /Image;
```

```
=>ext4load scsi 0:1 0x90100000 /xen.dtb;
```

```
=>ext4load scsi 0:1 0x93000000 /xen-ft2004-evm.efi;
```

```
=>fdt addr 0x90100000;
```

```
=>fdt set /chosen \#address-cells <1>;
```

```
=>fdt set /chosen \#size-cells <1>;
```

```
=>fdt set /chosen xen,xen-bootargs "console=dtuart dtuart=/soc/uart@28001000";
```

```
=>fdt set /chosen xen,dm0-bootargs "console=ttyAMA1,115200 root=/dev/sda2
rw";
```

```
=>fdt mknod /chosen module@0;
```

```
=>fdt set /chosen/module@0 compatible "xen,linux-zimage" "xen,multiboot-
module";
```

```
=>fdt set /chosen/module@0 reg <0x90200000 0x2000000>;
```

```
=>booti 0x93000000 - 0x90100000;
```

## 2、运行 Linux domu

(1) 创建配置文件 domu.cfg, 内容如下:

```
name="domu"
```

```
memory=512
```

```
kernel="/boot/Image"
```

```
ramdisk="/boot/xen-guest-image-minimal-ft2004-evm.cpio.gz"
```

```
extra="root=/dev/ram console=ttyAMA0 ramdisk_size=0x1000000"
```

(2) 创建 domu

```
xl create domu.cfg
```

(3) 查看现有的 domain

```
xl list
```

(4) 连接到 domu 的控制台，可通过“CTRL+]”退出 domu 的控制台

xl console domu

(5) 销毁 domain

xl destroy domu

Phytium 飞腾

## 7 系统镜像定制

### 7.1 内核定制

\$ bitbake -c menuconfig virtual/kernel (进入 kernel 选项配置)

根据需要可添加或删减内核组件。

### 7.2 文件系统定制

● 在例子文件系统中添加软件包 -- 通过修改 build\_<board>/conf/local.conf 配置文件添加软件包到镜像文件中:

```
IMAGE_INSTALL_append = " python"
```

```
PREFERRED_VERSION_python = " 3.4.0" 也可指定软件包版本
```

● 定制用户自己的文件系统 -- 在 sources/meta-phytium/meta-bsp/recipes-core/ 文件夹下执行下列命令 (以 custom 名字为例):

```
$ mkdir -p meta-phytium/meta-bsp/recipes-core/custom
```

```
$ cd meta-phytium/meta-bsp/recipes-core/custom
```

```
$ vi custom.bb
```

添加下面内容到 custom.bb 文件中:

```
IMAGE_INSTALL = "packagegroup-core-x11-base package1 package2"
```

```
inherit core-image
```

添加完成后, 可遵循第 5 节进入相应目录进行编译:

```
$ bitbake custom
```

● 编写新的应用软件包 (以下以 hello 名字为例, recipe 涉及概念较多, 下面只是简单介绍, 请具体参考官方文档来编写)

1、在 meta-phytium/meta-bsp/recipes-core/ 目录下创建 hello 文件夹。

2、在 hello 文件夹下创建 hello.bb 文件及 hello 文件夹。

3、在 hello 文件夹中放入 hello.c 及 Makefile 文件, hello 文件夹下目录文件如下:

```
[jieyun@localhost hello]$ tree .
.
├── hello
│   ├── hello.c
│   └── Makefile
└── hello.bb
```

图 7.1 hello 文件夹目录结构

4、hello.bb 文件内容如下 (也可以参考其他已有的 bb 文件):

```
SUMMARY = "Simple hello application"
```

```

SECTION = "libs"
LICENSE="MIT"
LIC_FILES_CHKSUM="file://hello.c;md5=0835ade698e0bcf8506ecdaf302"
SRC_URI = "\
file://hello.c \
file://Makefile
"
S = "${WORKDIR}"
do_compile() {
    make
}
do_install() {
    install -d ${D}${bindir}
    install -m 0755 ${S}/hello ${D}${bindir}/
}

```

上述 md5 值可在 ubunut 中运行 `md5sum hello.c` 来生成。

5、编译 hello:

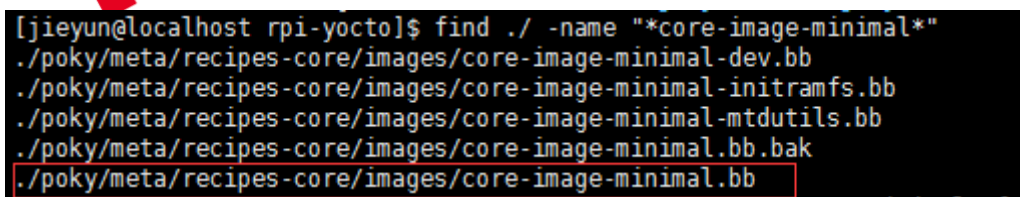
`$bitbake hello`

6、将 hello 添加到镜像中:

以添加到 `core-image-minimal` 中为例

查找 `core-image-minimal` 的 bb 文件位置

`$find / -name "*core-image-minimal*"`



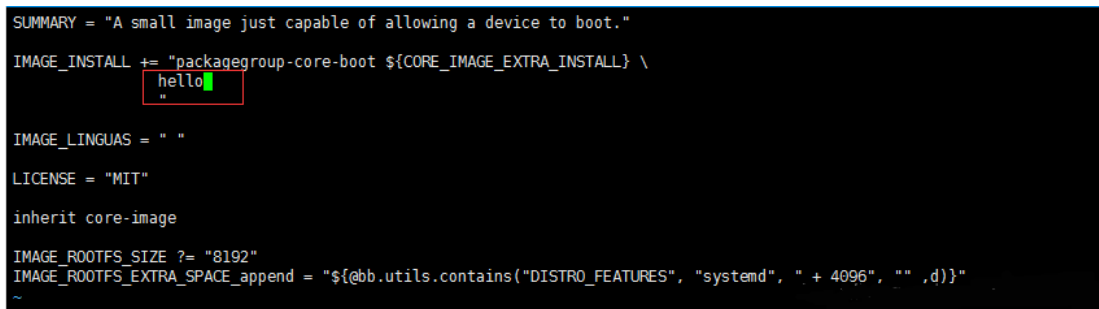
```

[jieyun@localhost rpi-yocto]$ find ./ -name "*core-image-minimal*"
./poky/meta/recipes-core/images/core-image-minimal-dev.bb
./poky/meta/recipes-core/images/core-image-minimal-initramfs.bb
./poky/meta/recipes-core/images/core-image-minimal-mtdutils.bb
./poky/meta/recipes-core/images/core-image-minimal.bb.bak
./poky/meta/recipes-core/images/core-image-minimal.bb

```

图 7.2 查找 core-image-minimal.bb

编辑 `core-image-minimal.bb` 文件，在 `IMAGE_INSTALL` 后面添加程序 `hello`



```

SUMMARY = "A small image just capable of allowing a device to boot."
IMAGE_INSTALL += "packagegroup-core-boot ${CORE_IMAGE_EXTRA_INSTALL} \'
hello
"
IMAGE_LINGUAS = " "
LICENSE = "MIT"
inherit core-image

IMAGE_ROOTFS_SIZE ?= "8192"
IMAGE_ROOTFS_EXTRA_SPACE_append = "${@bb.utils.contains("DISTRO_FEATURES", "systemd", " + 4096", "" ,d)}"

```

图 7.3 编辑 core-image-minimal.bb

重新编译镜像 bitbake core-image-minimal, 编译成功后启动, hello 程序就可以正常启动了。

具体实现可参考官方文档:

<https://www.yoctoproject.org/docs/3.1.2/mega-manual/mega-manual.html#new-recipe-writing-a-new-recipe>

## 7.3 BSP 定制

用户添加自己的开发板到 Yocto, 遵循下面的步骤, 以 FT2004 为例:

- 添加开发板配置

复制 ft2004-evm.conf (在 sources/meta-phytium/meta-bsp/conf/machine) 文件夹下, 重命名为 <custom-board>.conf (用户自定义开发板名字)

- 如需添加用户自己的内核补丁 kernel.patch (请根据实际名修改), 复制 kernel.patch 到 sources/meta-phytium/meta-bsp/recipes-kernel/linux/linux-phytium 文件夹下在 sources/meta-phytium/meta-bsp/recipes-kernel/linux/linux-phytium.inc 文件中添加:

```
SRC_URI_append_<custom-board>= " file://kernel.patch"
```

- 添加后设置环境变量

setup-env -m custom-board (用户自定义的开发板名字), 后续的编译操作等请参考第 4.3 节。

- Yocto 官方文档:

<https://www.yoctoproject.org/docs/3.1.2/dev-manual/dev-manual.html#platdev-newmachine>

## 7.4 软件包管理

默认包管理是 rpm, 也可以使用 debain 和 opkg 软件包管理:

- Dnf: RPM 软件包管理工具

添加下列变量 conf/local.conf

```
PACKAGE_CLASSES = "package_rpm"
```

```
EXTRA_IMAGE_FEATURES += "package-management"
```

- OPKG 软件包管理:

添加下列变量 conf/local.conf

```
PACKAGE_CLASSES = "package_ipk"
```

```
EXTRA_IMAGE_FEATURES += "package-management"
```

- APT: deb 软件包管理

添加下列变量 conf/local.conf



```
PACKAGE_CLASSES = "package_deb"
```

```
EXTRA_IMAGE_FEATURES += "package-management"
```

**Phytium飞腾**

## 8 常用软件

### 8.1 Docker 使用

Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的镜像中，然后发布到任何流行的 Linux 或 Windows 机器上，也可以实现虚拟化，飞腾嵌入 Linux 中 qt5 文件系统中默认包含了 docker。


- 查看 docker 是否可用

运行 docker 需要 root 或者 sudo 权限，在 root 用户下运行 **docker info** 查看 docker 程序是否存在，功能是否正常。

- 搜索容器

可以使用 **docker search** 命令搜索 docker hub 上公共的可用镜像，如需要查找 busybox 相关的镜像，可用如下命令：

#### docker search busybox



NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
busybox	Busybox base image.	1963	[OK]	
progrum/busybox		71		[OK]
radial/busyboxplus	Full-chain, Internet enabled, busybox made f00...	32		[OK]
yauritux/busybox-curl	Busybox with CURL	10		
arm32v7/busybox	Busybox base image.	8		
armhf/busybox	Busybox base image.	6		
odise/busybox-curl		4		[OK]
arm64v8/busybox	Busybox base image.	3		
prom/busybox	Prometheus Busybox Docker base images	2		[OK]
s390x/busybox	Busybox base image.	2		
arm32v6/busybox	Busybox base image.	2		
p7ppc64/busybox	Busybox base image for ppc64.	2		
aarch64/busybox	Busybox base image.	2		

图 8.1 搜索容器

结果如上图所示，之后可以使用 **docker pull** 拉取所需镜像。

- 拉取容器

根据上节搜索结果，拉取 **aarch64/busybox** 镜像到本地，命令如下所示，拉取成功后如下图。

#### docker pull aarch64/busybox

```
root@ft2004-evm:~# docker pull aarch64/busybox
Using default tag: latest
latest: Pulling from aarch64/busybox
a281075b7e6c: Pull complete
Digest: sha256:7656fa78c6c043b5f988d7effa6a04cf033cbd6ee24f6fd401214cabf0cbfb99
Status: Downloaded newer image for aarch64/busybox:latest
docker.io/aarch64/busybox:latest
root@ft2004-evm:~#
```

图 8.2 拉取容器

- 运行容器

**docker run -i -t aarch64/busybox**

上述命令运行一个基于 **aarch64** 架构的 **busybox** 的容器，**-i** 参数表明容器的标准输入是开启的，**-t** 参数为容器分配一个伪 **tty** 终端，容器创建完毕后出现如下所示的交互式终端，可以在容器中运行各种命令，使用 **exit** 退出容器。

```

root@ft2004-evm:~# docker run -it aarch64/busybox
/ # ls
bin    dev    etc    home   lib    lib64  proc   root   sys    tmp    usr    var
/ # cat /proc/cmdline
BOOT_IMAGE=/boot/Image console=ttyAMA1,115200 root=/dev/sda6
/ # exit
root@ft2004-evm:~#

```

图 8.3 运行容器

- 查看当前系统运行过的容器（正在运行或者已经退出）

```
docker ps -a
```

```

root@ft2004-evm:~# docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
7c4998ac410c   aarch64/busybox "/bin/bash"             52 minutes ago Created                                vigilant_zhukovsky
3a50b689b1fe   aarch64/busybox "sh"                    53 minutes ago Exited (0) 53 minutes ago          festive_wozniak
943a328b5ce7   aarch64/busybox "sh"                    54 minutes ago Exited (0) 53 minutes ago          angry_wing
aac8ac5ca799   aarch64/busybox "sh"                    54 minutes ago Exited (0) 54 minutes ago          elegant_robinson
a54216417ff1   aarch64/busybox "sh"                    56 minutes ago Exited (0) 55 minutes ago          busy_kirch
98f5779b4f59   aarch64/busybox "sh"                    58 minutes ago Exited (0) 58 minutes ago          serene_meitner
760c98c461e9   aarch64/busybox "/bin/bash"             58 minutes ago Created                                relaxed_herschel

```

图 8.4 查看当前系统运行过的容器

- 删除指定容器

```
docker rm 4030253341a5
```

```

root@ft2004-evm:~# docker rm 4030253341a5
4030253341a5
root@ft2004-evm:~# docker

```

图 8.5 删除指定容器

- 查看当前系统的容器镜像

可用 `docker images` 查看当前系统的容器镜像，命令和结果如下所示。

```
docker images
```

```

root@ft2004-evm:~# docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
aarch64/busybox latest    27725c36cdc8   3 years ago    3.31MB
root@ft2004-evm:~#

```

图 8.6 查看当前系统的容器镜像

- 删除当前系统指定镜像

```
docker rmi aarch64/busybox
```

```

root@ft2004-evm:~# docker rmi aarch64/busybox
Untagged: aarch64/busybox:latest
Untagged: aarch64/busybox@sha256:7656fa78c6c043b5f988d7effa6a04cf033cbd6ee24f6fd401214cabf0cbfb99
Deleted: sha256:27725c36cdc8dd631fec82926947bf4be6799f049a329c7cca9800fddf7ff9d2
Deleted: sha256:f91599b3986be816a2c74a3c05fda82e1b29f55eb12bc2b54fbdfc3b5773edc
root@ft2004-evm:~#

```

图 8.7 删除当前系统指定镜像

## 8.2 实时内核测试

为了测试 RT 调度器的性能，需将内核替换为 Image-rt，该内核完全开启 RT 选项。测试过程中主要使用 `stress` 和其他一些压力测试工具增加系统，使用 `cyclicttest` 查看系统实时性能，用 `upload` 查看系统负载。

- `cyclicttest -p 80 -t 5 -n`

使用上述命令查看系统实时性能，-p 指定调度优先级，-t 表示创建的线程数目，-n 表示一直测试，手动 Ctrl-C 停止测试。

- stress -c 8

上述命令增加系统负载，-c 表明需要增加的系统负载数目，一般一个单核满载为 1，因此对于一个四核 CPU（无超线程）而言，系统满载为 4，表明 CPU 一直运行，没有任何空闲；系统负载为越高，系统压力越大，越考验调度器的性能。

如下所示为 cyclicttest 的测试结果，T 表示线程号，P 表示调度优先级，I 表示时间间隔（单位微秒），C 表示被调度次数，之后四行分别是最小、最近、平均、最大的延时，单位为微秒。

```
T: 0 ( 2588) P:80 I:1000 C: 64705 Min:      2 Act:      7 Avg:      6 Max:      17
T: 1 ( 2589) P:80 I:1500 C: 43137 Min:      2 Act:      9 Avg:      6 Max:      16
T: 2 ( 2590) P:80 I:2000 C: 32352 Min:      2 Act:      5 Avg:      4 Max:      11
T: 3 ( 2591) P:80 I:2500 C: 25882 Min:      2 Act:      6 Avg:      5 Max:      14
T: 4 ( 2592) P:80 I:3000 C: 21568 Min:      2 Act:      6 Avg:      4 Max:      10
```

图 8.8 cyclicttest 的测试结果

## 8.3 图形和显式

core-image-xfce 和 core-image-weston 文件系统，默认不支持 X100 gpu。

如果开发板支持 X100 gpu，修改 meta-phytium/meta-bsp/conf/machine/phytium-base.inc 文件，添加 gpu 功能重新编译文件系统：

```
MACHINE_FEATURES ?= "pci ext2 ext3 serial usbhost alsa touchscreen
keyboard gpu"
```

- X11: core-image-xfce 文件系统,支持 X11 协议.通过 glmark2 和 glxgears 测试 gpu 性能。

- Wayland/Weston: core-image-weston 文件系统支持 Wayland 协议。

- QT 图形: core-image-xfce 文件系统里,测试用例在/usr/share/examples。core-image-weston 文件系统里,测试用例在/usr/share/examples/wayland。

## 8.4 多媒体

core-image-xfce 和 core-image-weston 文件系统，默认不支持 vpu。

如果开发板支持 vpu，

(1) 修改 meta-phytium/meta-bsp/conf/machine/phytium-base.inc 文件，添加 vpu 功能：

```
MACHINE_FEATURES ?= "pci ext2 ext3 serial usbhost alsa touchscreen
keyboard vpu"
```

(2) 修改 meta-phytium/meta-bsp/conf/layer.conf 文件，去掉：

```
BBMASK += "meta-bsp/recipes-graphics/vpu/*"
```

然后重新编译文件系统。

文件系统里显示 CST-OMX 提供的 GStreamer 信息：

```
$ gst-inspect-1.0 | grep omx
```

```
omx:  omxmjpegdec: OpenMAX MJPEG Video Decoder
```

```
omx:  omxh263dec: OpenMAX H.263 Video Decoder
```

```
omx:  omxmpeg2videodec: OpenMAX MPEG2 Video Decoder
```

```
omx:  omxmpeg4videodec: OpenMAX MPEG4 Video Decoder
```

```
omx:  omxh265dec: OpenMAX-IMG HEVC Video Decoder
```

```
omx:  omxh264dec: OpenMAX H.264 Video Decoder
```

用 gstreamer playbin 播放音频视频文件：

```
$ gst-launch-1.0 -v playbin uri=file:///<video>.mp4 video-sink=glimagesink
```

用 gst-launch 构造管道：

```
$ gst-launch-1.0 -v filesrc location=<video>.mp4 ! qtdemux ! h264parse !  
omxh264dec ! glimagesink
```

## 9 附录

### 9.1 常见问题

#### 1、支持调试

为了支持调试，可以添加下列规则到 local.conf 文件中：

```
IMAGE_FEATURES_append = " tools-debug dbg-pkgs"
```

#### 2、更改内核组件选项

运行下列命令：

```
$ bitbake -c menuconfig virtual/kernel
```

```
$ bitbake -c compile virtual/kernel (只运行编译任务)
```

#### 3、内核源码位置：

```
build_ft2004-evm/tmp/work/ft2004-evm-phy-linux/linux-phytium/4.19-r0/git
```

#### 4、重新编译软件包

用户重新编译软件包，运行下列命令：

```
$ bitbake -c cleansstate < package name>
```

清除编译的输出结果和 shared state cache

```
$ bitbake < package name>
```

编译 package

#### 5、编译生成的镜像位置：

```
tmp/deploy/images/ft2004-evm/。
```

#### 6、qt x11 去掉桌面

添加下列变量到 local.conf

```
IMAGE_FEATURES_remove = "x11-sato"
```

```
RDEPENDS_${PN}_remove = "matchbox-terminal"
```

#### 7、内核打补丁

添加用户自己的内核补丁 kernel.patch（请根据实际名修改），复制 kernel.patch 到 sources/meta-phytium/meta-bsp/recipes-kernel/linux/linux-phytium 文件夹下。

需要修改 sources/meta-phytium/meta-bsp/recipes-kernel/linux/linux-phytium.inc 文件。

如果补丁是针对指定开发板用如下规则：

```
SRC_URI_append_<custom-board>= " file://kernel.patch"
```

如果补丁适用所有开发板用如下规则：

```
SRC_URI_append = " file://kernel.patch"
```

#### 8、编译可扩展 SDK

添加下列变量到 `build_<board>/conf/local.conf`

```
SDK_INCLUDE_BUILDTOOLS = ""
```

然后运行:

```
bitbake -c populate_sdk_ext core-image-xfce
```

## 9.2 编译环境配置

交叉编译是在一个平台生成另一个平台的可执行代码，目前常见的是在 x86 平台生成其他平台的代码，如 x86 下生成 arm64 的二进制文件，目前飞腾支持基于 Yocto 和 Linaro 的交叉编译工具链。

### 9.2.1 基于 Yocto 生成交叉编译工具链

飞腾提供的 SDK 包可构建基于 Yocto 的交叉编译工具链，构建安装步骤如下所示。

- 进入飞腾 SDK 包目录，配置环境变量: `. setup-env -m <machine>`
- 编译交叉工具链: `bitbake meta-toolchain`
- 编译完成后进入构建目录下的 `tmp/deploy/sdk`
- 切换到 root 权限，运行 `phytium-glibc-x86_64-meta-toolchain-aarch64-toolchain-<version>.sh`，根据提示安装交叉编译器，一般选默认配置即可。
- 假设交叉编译器安装目录为 `/opt`，使用前按照如下命令配置即可使用。

```
$ source /opt/poky/<version>/environment-setup-aarch64-poky-linux
```

```
$ cd <linux-source path>
```

```
$ make defconfig sdk.config
```

```
$ make
```

### 9.2.2 下载 Linaro 的交叉编译工具链

Linaro 是一家非营利性质的开源软件公司，自成立以来一直致力于推动基于 ARM 的开源软件开发，提供多种编译工具链，为基于 Linux 内核的开发者提供了稳定得编译环境。基于 Linaro 的交叉编译工具链可从 Linaro 官网下载，以下截止到 2020 年 7 月时最新的一个交叉编译工具。

[https://releases.linaro.org/components/toolchain/binaries/latest-7/aarch64-linux-gnu/gcc-linaro-7.5.0-2019.12-x86\\_64\\_aarch64-linux-gnu.tar.xz](https://releases.linaro.org/components/toolchain/binaries/latest-7/aarch64-linux-gnu/gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu.tar.xz)

配置步骤:

- 解压到 `/opt` 目录

```
$ tar -xf gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu.tar.gz -C /opt
```

- 配置环境变量和编译内核



```
$ export PATH=/opt/gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu/bin:
$PATH
$ export ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-
$ export CC=aarch64-linux-gnu-gcc
$ cd <linux-source path>
$ make defconfig sdk.config
$ make
```

### 9.3 已知缺陷

- UEFI 固件致的启动卡死问题

使用 UEFI 固件启动，同时 Linux 内核开启 CONFIG\_CPU\_FREQ 和 ARM\_SCPI\_PROTOCOL 这两个选项的情况下，系统有一定概率会在启动过程中卡死（大约 60 次会复现一次）。目前可通过以下两个方法规避该问题：

使用设备树引导内核启动。

使用 UEFI 引导内核启动，可以配置内核关闭 CONFIG\_CPU\_FREQ 和 ARM\_SCPI\_PROTOCOL 这两个选项。

注：UEFI 固件更新后可彻底解决该问题。

- 使用 X100 gpu 时，media player 无法播放音频文件

使用 X100 gpu，在桌面环境下点击 media player，选择音频文件播放时，桌面会黑屏重启。目前可通过命令行 gst-play-1.0 播放音频，如 gst-play-1.0 wavtest.wav。

### 9.4 参考

<https://www.yoctoproject.org/docs/3.1/brief-yoctoprojectqs/brief-yoctoprojectqs.html>

### 9.5 支持外设列表

表 9-1 FT2000/4 和 FT2000/2 的 Phytium Embedded Linux V1.5

CPU(loader) peripheral	FT2000/4 (Uboot)	FT2000/4 (UEFI)	FT2000/2 (Uboot)
DIMM DDR4	√	√	√
GbE	√	√	√
UART	√	√	√
SD Card	√	√	
HDAudio	√	√	



M.2 SSD	✓	✓	
USB	✓	✓	✓
PCIE	✓	✓	✓
I2C	✓	✓	✓
CAN	✓		
SPI	✓		
QSPI	✓		
GPIO	✓	✓	✓
WDT	✓	✓	
Temp Sensor	✓	✓	

表 9-2 D2000 的 Phytium Embedded Linux V1.5

peripheral \ CPU(loader)	D2000 (Uboot)	D2000 (UEFI)
DIMM DDR4	✓	✓
GbE	✓	✓
UART	✓	✓
SD Card	✓	✓
HDAudio	✓	✓
M.2 SSD	✓	✓
USB	✓	✓
PCIE	✓	✓
I2C	✓	✓
CAN	✓	
SPI	✓	
QSPI	✓	
GPIO	✓	✓
WDT		
Temp Sensor	✓	✓