

# 飞腾嵌入式 Linux Yocto 4.0 用户手册 ( V2.0 )

飞腾信息技术有限公司

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

2023 年 05 月 15 日

版权所有 © 飞腾信息技术有限公司 2022。保留一切权利。

未经本公司同意，任何单位、公司或个人不得擅自复制、翻译、摘抄本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

### 商标声明

Phytium 和其他飞腾商标均为飞腾信息技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

### 特别提示

本文档仅作为使用指导，飞腾对本文档内容不做任何明示或暗示的声明或保证。本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

由于产品版本升级或其他原因，本文档内容会不定期进行更新，如有变更，恕不另行通知。

### 最新技术资源

飞腾嵌入式软件开源社区 [https://gitee.com/phytium\\_embedded](https://gitee.com/phytium_embedded)

或者访问飞腾软件开发者平台 <https://service.phytium.com.cn/developer/>

## 目录

1 概述 .....	1
2 软硬件特性 .....	1
3 开发环境配置 .....	2
3.1 硬件配置 .....	2
3.2 开发软件安装 .....	2
3.2.1 repo 方式 .....	3
3.2.2 ISO 方式 .....	3
4 系统镜像构建 .....	4
4.1 设置开发板环境变量 .....	4
4.2 执行构建 .....	4
4.2.1 最小系统和 xfce 桌面系统 .....	4
4.2.2 weston 系统 .....	4
4.2.3 无网络环境 .....	5
4.2.4 更换内核版本 .....	5
4.2.5 实时内核 .....	5
4.2.6 支持 VPU .....	5
4.2.7 Xenomai 系统 .....	6
5 系统镜像运行 .....	7
5.1 Uboot 启动 .....	7
5.2 UEFI 启动 .....	8
6 安装盘 .....	12
6.1 ISO 编译 .....	12
6.2 安装盘制作 .....	12
6.3 系统安装 .....	13
6.4 系统启动 .....	14
7 系统镜像定制 .....	14
7.1 内核定制 .....	14
7.2 文件系统定制 .....	15

7.3 BSP 定制 .....	17
7.4 软件包管理 .....	17
8 常用软件 .....	18
8.1 Docker 使用 .....	18
8.2 图形和显式 .....	20
8.3 多媒体 .....	20
9 附录 .....	21
9.1 常见问题 .....	21
9.2 编译环境配置 .....	22
9.2.1 基于 Yocto 生成交叉编译工具链 .....	22
9.2.2 下载 Linaro 的交叉编译工具链 .....	23
9.3 参考 .....	23
A 更新记录 .....	24

## 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者提供技术支持，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 4.0 工程如何构建针对飞腾 E2000 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 4.0 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

飞腾 E2000 Embedded Linux 的主要功能如下：

- a) 工作于 ARM v8 AARCH64 模式
- b) 支持 E2000Q、E2000D、E2000S
- c) 支持飞腾 E2000 参考板（demo 板）、多款行业开发板（COMe、电力行业板、Mini-ITX、VPX、教育开发板）
- d) 基于飞腾 Linux kernel( [https://gitee.com/phytium\\_embedded/phytium-linux-kernel.git](https://gitee.com/phytium_embedded/phytium-linux-kernel.git) )
- e) 支持 Linux 实时补丁
- f) 支持 VPU
- g) 支持 Yocto 4.0.6
- h) 支持 Mesa 22.0.3
- i) 支持 UEFI 和 Uboot 启动
- j) 支持 Qt 5.15.3
- k) 支持 Qt OpenGL（基于 X11）
- l) 支持 Yocto 桌面
- m) 支持 Docker-ce 20.10.21

飞腾 E2000D 参考板硬件平台框图如图 2.1 所示:

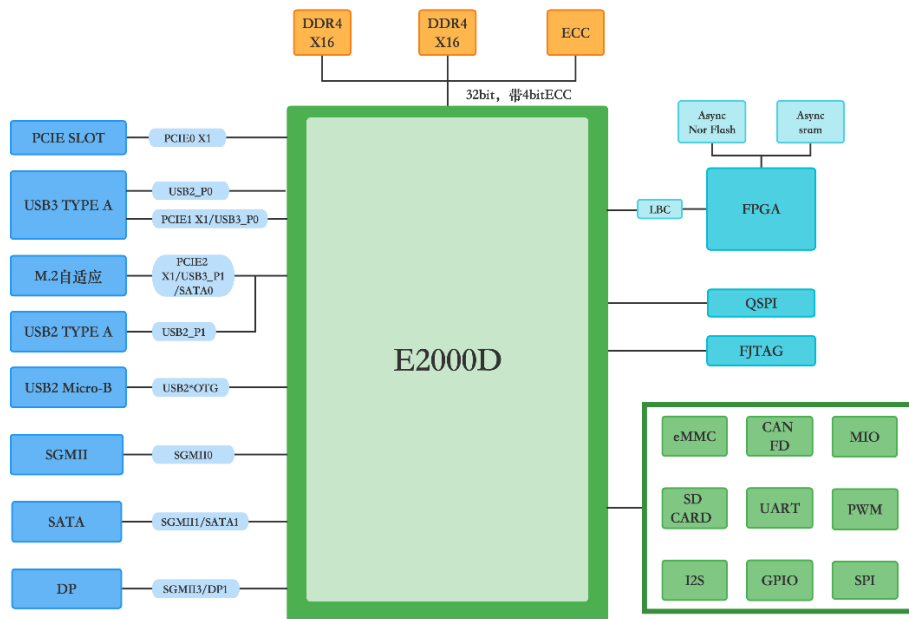


图 2.1 E2000D 参考板硬件平台框图

## 3 开发环境配置

### 3.1 硬件配置

- a) 最少 4GB 内存
- b) Ubuntu20.04、Ubuntu22.04、Debian11 系统之一
- c) 磁盘剩余空间至少 50GB

### 3.2 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 libcursesw5-dev openssl libssl-dev zstd
```

飞腾嵌入式 Linux 发行版以 repo 和 ISO 两种形式发布。

### 3.2.1 repo 方式

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
```

下载 Yocto 层

```
$ export PATH=${PATH}:~/bin
$ mkdir <phytium-path>
$ cd <phytium-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-linux-yocto.git -b master
$ repo sync --force-sync
$ source setup-env -m <machine> ( Supported machines: e2000 )
```

### 3.2.2 ISO 方式

包含以下两个文件:

表 3-1 发布的 ISO 文件

phytium-linux-yocto-<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-linux-yocto-<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

从源码编译构建系统需要安装以下的 ISO:

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

选择安装 cache ISO 以加快编译速度:

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

## 4 系统镜像构建

可构建的镜像为：

表 4-1 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

下面介绍如何构建系统镜像。

（注意： 请不要用 root 用户执行下列操作）

### 4.1 设置开发板环境变量

```
$ cd phytium-linux-yocto-<version>-<yyyymmdd>-yocto or <phytium-path>  
$ source setup-env -m <machine> （ Supported machines: e2000， 设置 E2000 开发板  
的环境变量 ）
```

### 4.2 执行构建

以下使用<rootfs\_image>表示可选系统镜像的名字。

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

#### 4.2.1 最小系统和 xfce 桌面系统

编译系统镜像：

```
$ bitbake <rootfs_image>
```

<rootfs\_image>表示 core-image-minimal 或 core-image-xfce

#### 4.2.2 weston 系统

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"
```

编译系统镜像：

```
$ bitbake core-image-weston
```



### 4.2.3 无网络环境

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

```
BB_NO_NETWORK = "1"
```

编译系统镜像:

```
$ bitbake <rootfs_image>
```

### 4.2.4 更换内核版本

飞腾 Linux kernel ( [https://gitee.com/phytium\\_embedded/phytium-linux-kernel.git](https://gitee.com/phytium_embedded/phytium-linux-kernel.git) ) 中除了 master 分支外, 有四个分支: linux-5.10, linux-5.10-rt, linux-4.19, linux-4.19-rt. yocto 编译系统镜像时默认编译 Linux 5.10 内核, 如果需要将内核版本修改为 Linux 4.19, 要在 conf/local.conf 文件中添加:

```
PREFERRED_VERSION_linux-phytium = "4.19"
```

编译内核:

```
$ bitbake virtual/kernel
```

编译系统镜像:

```
$ bitbake <rootfs_image>
```

### 4.2.5 实时内核

如果编译带 Linux 实时补丁 (RT) 的系统镜像, 需要在 conf/local.conf 文件中添加下列变量:

- 1) 如果编译 Linux 5.10 RT 内核, 添加:

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

- 2) 如果编译 Linux 4.19 RT 内核, 添加:

```
PREFERRED_PROVIDER_virtual/kernel = "linux-phytium-rt"  
PREFERRED_VERSION_linux-phytium-rt = "4.19"
```

编译内核:

```
$ bitbake virtual/kernel
```

编译系统镜像:

```
$ bitbake <rootfs_image>
```

### 4.2.6 支持 VPU

core-image-xfce 和 core-image-weston 文件系统, 默认支持 vpu (视频解码器)。

编译系统镜像:

```
$ bitbake <rootfs_image>
```

#### 4.2.7 Xenomai 系统

##### 1) 编译 Xenomai cobalt 模式

使用 xenomai 内核 ( [https://gitee.com/phytium\\_embedded/linux-kernel-xenomai.git](https://gitee.com/phytium_embedded/linux-kernel-xenomai.git) ), 该内核有 Linux 5.10 和 Linux 4.19 两个版本。

a) 如果编译 Linux 5.10 版本 xenomai 内核, 需要在 conf/local.conf 文件中添加:

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

b) 如果编译 Linux 4.19 版本 xenomai 内核, 需要在 conf/local.conf 文件中添加:

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

```
PREFERRED_VERSION_linux-xenomai-phytium = "4.19"
```

```
PREFERRED_VERSION_xenomai = "3.1.3"
```

编译 core-image-rt 文件系统镜像:

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

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

使用实时内核 ( [https://gitee.com/phytium\\_embedded/phytium-linux-kernel.git](https://gitee.com/phytium_embedded/phytium-linux-kernel.git) ), 该仓库中有 linux-5.10-rt 和 linux-4.19-rt 分支。

a) 如果编译 Linux 5.10 RT 内核:

修改 conf/local.conf 文件, 添加:

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

修改 sources/meta-phytium/meta-xenomai/recipes-xenomai/xenomai/xenomai\_3.2.2.bb, 将

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

修改为

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

b) 如果编译 Linux 4.19 RT 内核:

修改 conf/local.conf 文件, 添加:

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

```
PREFERRED_VERSION_linux-phytium-rt = "4.19"
```

```
PREFERRED_VERSION_xenomai = "3.1.3"
```

修改 sources/meta-phytium/meta-xenomai/recipes-xenomai/xenomai/xenomai\_3.1.3.bb, 将

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

修改为

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

编译 core-image-rt 文件系统镜像

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

## 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

使用 U 盘、SATA 盘或 NVME 盘，可启动表 4-1 中的所有文件系统，以 core-image-minimal 文件系统为例。

**前提条件：**

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

**在主机端将系统镜像拷贝到 USB、SATA 或 NVME 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。

将设备分区：

```
$ 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，并将系统镜像拷贝到设备：

```
$ 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 e2000d-demo-board.dtb /mnt ( 以 e2000d demo 板为例, 其它开发板修改为对应的设备树即可 )
$ 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
```

#### 在开发板上设置 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 audit=0 earlycon=pl011,0x2800d000
root=/dev/sda2 rw
=>usb start
=>ext4load usb 0:1 0x90100000 Image
=>ext4load usb 0:1 0x90000000 e2000d-demo-board.dtb
=>booti 0x90100000 - 0x90000000
```

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 ):

```
=>setenv bootargs console=ttyAMA1,115200 audit=0 earlycon=pl011,0x2800d000
root=/dev/sda2 rw
=>ext4load scsi 0:1 0x90100000 Image
=>ext4load scsi 0:1 0x90000000 e2000d-demo-board.dtb
=>booti 0x90100000 - 0x90000000
```

文件系统在 M.2 接口的 NVME 盘上:

```
=>setenv bootargs console=ttyAMA1,115200 audit=0 earlycon=pl011,0x2800d000
root=/dev/nvme0n1p2 rw
=>ext4load nvme 0:1 0x90100000 Image
=>ext4load nvme 0:1 0x90000000 e2000d-demo-board.dtb
=>booti 0x90100000 - 0x90000000
```

## 5.2 UEFI 启动

基于 E2000 的开发板, 支持 Uboot 和 UEFI 两种引导方式, 本节将说明 UEFI 的启动方法。

#### 前提条件

- a) 一台基于 Linux 发行版的主机。

- b) 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）。

### 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@sxf-OptiPlex-7070:/home/sxf# mkfs.vfat /dev/sdb1
mkfs.fat 4.1 (2017-01-24)
root@sxf-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@sxf-OptiPlex-7070:/home/sxf#
```

图 5.7 格式化分区

- 10) 挂载根文件系统分区, 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, 示例如下:

```
$ sudo mount /dev/sdb1 /mnt
$ sudo mount /dev/sdb2 /mnt2
$ cd <image-path> ( <image-path>为 tmp/deploy/images/<machine> )
$ sudo tar zxvf core-image-minimal.tar.gz -C /mnt2
$ sudo cp -rL /mnt2/boot/* /mnt
$ sudo cp Image /mnt
```

修改引导文件/`/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
}
```

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)。

可根据需要仿照上述格式添加多个表项。

卸载分区:



```
$ sync  
$ 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。

## 6 安装盘

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编译成 ISO，制作成安装盘并安装到硬盘中，供用户快速测试使用。

### 6.1 ISO 编译

- 1) 设置开发板环境变量

```
$ cd phytium-linux-yocto-<version>-<yyyymmdd>-yocto or <phytium-path>  
$ source setup-env -m <machine> ( Supported machines: e2000 )
```

- 2) 将以下变量添加到 conf/local.conf 中

```
DISTRO_FEATURES:append = " anaconda-support pam"  
VIRTUAL-RUNTIME_init_manager = "systemd"  
DISTRO_FEATURES:append = " systemd"  
DISTRO_FEATURES_BACKFILL_CONSIDERED:append = " sysvinit"
```

- 3) 编译系统

```
$ bitbake core-image-xfce
```

- 4) xfce 系统编译完成后，在 conf/local.conf 中把上面变量去掉，把下面这些变量添加进去：

```
DISTRO = "anaconda"  
INSTALLER_TARGET_BUILD = "<phytium-  
path>/build_e2000/tmp/deploy/images/e2000/core-image-xfce-e2000.ext4"  
INSTALLER_TARGET_IMAGE = "core-image-xfce"
```

- 5) 编译 ISO

```
$ bitbake core-image-anaconda
```

编译完成后，ISO 位于 build\_e2000/tmp-glibc/deploy/images/e2000/core-image-anaconda-e2000.iso

### 6.2 安装盘制作

使用上节编译的 ISO 文件制作安装盘，具体使用方法如下：

在主机端插入 U 盘，运行下列命令制作安装盘（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) 如果使用 Uboot 启动:

```
$ sudo mount -o loop core-image-anaconda-e2000.iso media/  
$ sudo mkfs.ext4 -L boot /dev/sdb1  
$ sudo mount /dev/sdb1 /mnt  
$ sudo rsync -a -P media/ /mnt/  
$ sudo umount /mnt
```

- 2) 如果使用 UEFI 启动:

```
$ sudo dd if=core-image-anaconda-e2000.iso of=/dev/sdb bs=1M
```

## 6.3 系统安装

将安装盘插入开发板的 USB 接口，启动。

- 1) Uboot 启动模式下，在串口控制台设置 Uboot 环境变量:

```
=>setenv bootargs root=/dev/ram0 rootdelay=5 rw initrd=0x93000000,140M  
=>usb start  
=>ext4load usb 0:1 0x90100000 Image  
=>ext4load usb 0:1 0x90000000 dtb/e2000d-demo-board.dtb (以 e2000d demo 板为例，其它开发板修改为对应的设备树即可)  
=>ext4load usb 0:1 0x93000000 initrd  
=>booti 0x90100000 - 0x90000000
```

- 2) UEFI 启动模式下，开机按 F2，选择 U 盘作为启动设备，然后在 GRUB 界面选择 “boot graphics console”，等待进入图形安装界面。

在图型安装界面，需要做以下设置:

首先需要选择安装过程使用的语言，保持默认即可，点击 Continue，来到 “INSTALLATION SUMMARY” 安装主界面，这里有三个地方需要设置。

- 选择系统的安装位置: 点击 Installation Destination，选择想要将系统安装到哪个设备 (USB 安装盘之外的磁盘，需要提前备份磁盘上的数据)，选择设备后，点击屏幕左上角的 Done，在弹出的界面中依次点击 Reclaim space -> Delete all -> Reclaim space，表示要删除该磁盘上的所有分区和数据，然后回到了安装主界面。
- 时区: 点击 Time & Date，选择 Region: Asia, City: Shanghai，点击 Done，回到安装主界面。
- 用户设置: 点击 Root Password，输入 root 用户的密码，点击 Done；点击 User Creation 创建新用户，输入用户名密码，点击 Done，回到安装主界面。

配置完成后，点击系统安装主界面右下角的 Begin Installation，开始安装。

安装过程如果提示写 bootloader 错误，请选 Yes 选项。安装成功后，点击界面右下角的 Reboot System，重启开发板。

## 6.4 系统启动

- 1) Uboot 启动模式下，串口控制台可以设置通过 SATA 盘、NVME 盘或 MMC 卡来启动系统。

如果前面选择将系统安装到 SATA 盘，则启动参数为：

```
=>setenv bootargs root=/dev/mapper/openembedded_e2000-root rootdelay=5 rw  
initrd=0x93000000,140M console=ttyAMA1,115200  
=>ext4load scsi 0:1 0x90100000 Image;  
=>ext4load scsi 0:1 0x90000000 e2000d-demo-board.dtb;  
=>ext4load scsi 0:1 0x93000000 initrd;  
=>booti 0x90100000 - 0x90000000
```

如果前面选择将系统安装到 NVME 盘，则启动参数为：

```
=>setenv bootargs root=/dev/mapper/openembedded_e2000-root rootdelay=5 rw  
initrd=0x93000000,140M console=ttyAMA1,115200  
=>ext4load nvme 0:1 0x90100000 Image;  
=>ext4load nvme 0:1 0x90000000 e2000d-demo-board.dtb;  
=>ext4load nvme 0:1 0x93000000 initrd;  
=>booti 0x90100000 - 0x90000000
```

如果前面选择将系统安装到 MMC 卡，则启动参数为：

```
=>setenv bootargs root=/dev/mapper/openembedded_e2000-root rootdelay=5 rw  
initrd=0x93000000,140M console=ttyAMA1,115200  
=>ext4load mmc 0:1 0x90100000 Image;  
=>ext4load mmc 0:1 0x90000000 e2000d-demo-board.dtb;  
=>ext4load mmc 0:1 0x93000000 initrd;  
=>booti 0x90100000 - 0x90000000
```

- 2) UEFI 启动模式下，重新启动即可进入刚才安装的系统。

## 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

- 1) 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" (可指定软件包版本)
```

- 2) 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:

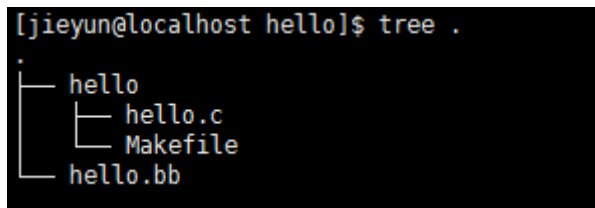
```
IMAGE_INSTALL = "packagegroup-core-x11-base package1 package2"
inherit core-image
```

添加完成后, 进入相应目录进行编译:

```
$ bitbake custom
```

- 3) 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)

- a) 在 meta-phytium/meta-bsp/recipes-core/ 目录下创建 hello 文件夹。
- b) 在 hello 文件夹下创建 hello.bb 文件及 hello 文件夹。
- c) 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

- d) 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= 0835ade698e0bcf8506ecda2f7b4f302 "
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 ubuntu 中运行 md5sum hello.c 来生成。

e) 编译 hello:

```
$ bitbake hello
```

f) 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://docs.yoctoproject.org/4.0.9/dev-manual/common-tasks.html#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 E2000 为例：

### 1) 添加开发板配置

复制 e2000.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）

2) 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"
```

### 3) 添加后设置环境变量

```
$ source 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 章。
```

Yocto 官方文档：

<https://docs.yoctoproject.org/4.0.9/ref-manual/index.html>

## 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：

### 1) Dnf：RPM 软件包管理工具

添加下列变量到 conf/local.conf

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

### 2) OPKG 软件包管理：

添加下列变量到 conf/local.conf

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

### 3) APT：deb 软件包管理

添加下列变量到 conf/local.conf

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

## 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。

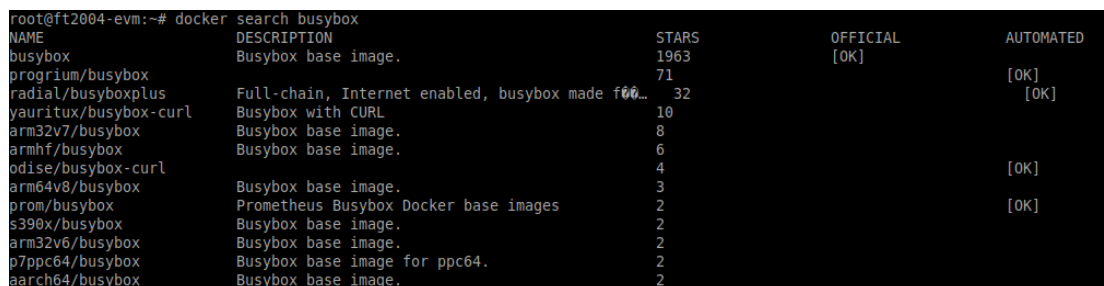
#### 1) 查看 docker 是否可用

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。

#### 2) 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]	
progrium/busybox		71		[OK]
radial/busyboxplus	Full-chain, Internet enabled, busybox made for...	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		
ppc64/busybox	Busybox base image for ppc64.	2		
aarch64/busybox	Busybox base image.	2		

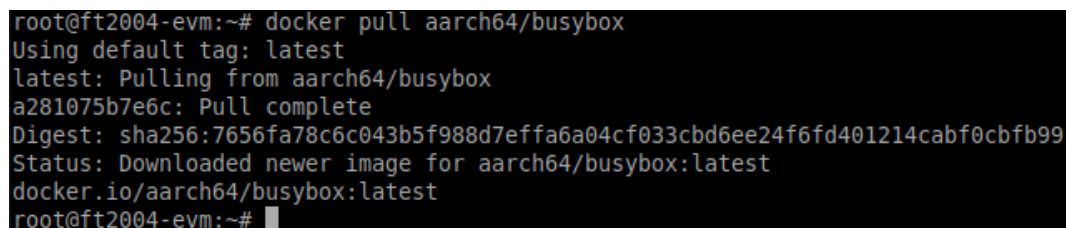
图 8.1 搜索容器

结果如上图所示，之后可以使用 docker pull 拉取所需镜像。

#### 3) 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

## 4) 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

## 5) 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
7a50b689b1fe	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
9ac8ac5ca799	aarch64/busybox	"sh"	54 minutes ago	Exited (0) 54 minutes ago		elegant_robinson
954216417ff1	aarch64/busybox	"sh"	56 minutes ago	Exited (0) 55 minutes ago		busy_kirch
90f5779b4f59	aarch64/busybox	"sh"	58 minutes ago	Exited (0) 58 minutes ago		serene_meltner
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

## 6) 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

## 7) 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

图 8.6 查看当前系统的容器镜像



## 8) 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
Deleted: 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 图形和显式

- 1) X11: core-image-xfce 文件系统, 支持 X11 协议。通过 glmark2 和 glxgears 测试 gpu 性能。
- 2) Wayland/Weston: core-image-weston 文件系统支持 Wayland 协议。
- 3) 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。

## 8.3 多媒体

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

core-image-xfce 和 core-image-weston 文件系统默认安装了 Gstreamer 来播放音频视频。

- 1) 文件系统里显示 GST-OMX 提供的 GStreamer 信息:

```
$ gst-inspect-1.0 | grep omx
omx: omxrealvideodec: OpenMAX-IMG REAL Video Decoder
omx: omxsorensodec: OpenMAX-IMG Sorenson Spark Video Decoder
omx: omxvp8dec: OpenMAX VP8 Video Decoder
omx: omxvp6dec: OpenMAX VP6 Video Decoder
omx: omxvc1dec: OpenMAX WMV Video Decoder
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
```

- 2) 用 gstreamer playbin 播放本地音频视频文件:

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

- 3) 用 gst-launch 构造管道, 通过 vpu 硬解码播放本地视频文件:

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



- 4) 通过 vpu 硬解码播放摄像头推送的网络流, 比如以 rtsp 协议传输、h264 格式编码的视频流:

```
$ gst-launch-1.0 rtspsrc location="<rtsp-address>" ! rtph264depay ! 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_e2000/tmp/work/e2000-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/e2000/
```

- 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 (请根据实际名称修改), 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) 编译应用开发标准 SDK

对于 core-image-xfce, 运行:

```
$ bitbake -c populate_sdk core-image-xfce
```

对于 core-image-weston:

添加下列变量到 build\_<board>/conf/local.conf :

```
DISTRO_FEATURES:remove = "x11"
```

然后运行:

```
$ bitbake -c populate_sdk core-image-weston
```

#### 9) 编译应用开发可扩展 SDK

对于 core-image-xfce:

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
```

对于 core-image-weston:

添加下列变量到 build\_<board>/conf/local.conf :

```
DISTRO_FEATURES:remove = "x11"
```

```
SDK_INCLUDE_BUILDTOOLS = ""
```

然后运行:

```
$ bitbake -c populate_sdk_ext core-image-weston
```

#### 10) 编译 qt5 交叉工具链

```
$ bitbake meta-toolchain-qt5
```

## 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。

a) 进入飞腾 SDK 包目录, 配置环境变量:

```
$ source setup-env -m <machine>
```

b) 编译交叉工具链:

```
$ bitbake meta-toolchain
```

c) 编译完成后进入构建目录下的 tmp/deploy/sdk

d) 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。

e) 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/phytium/<version>/environment-setup-aarch64-phy-linux
```

f) 编译 linux 内核：

```
$ cd <linux-source-path>
$ make e2000_defconfig
$ 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：

a) 解压到 /opt 目录

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

b) 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 e2000_defconfig
$ make
```

### 9.3 参考

<https://docs.yoctoproject.org/4.0.9/>

## A 更新记录

发布日期	版本	说明
2022-08-11	V1.0	初稿，针对飞腾 E2000 系列
2023-02-24	V1.1	支持构建 xenomai 系统；支持制作安装盘，使用安装盘安装系统；默认支持 vpu 及 vpu 的使用；支持 5.10 和 4.19 内核版本的更换
2023-03-24	V1.2	删除 ISO 发布的二进制安装包，修改安装盘安装系统后的启动参数，增加开发环境的软件依赖包
2023-05-15	V2.0	Yocto 升级到 4.0，文件名改为“飞腾嵌入式 Linux Yocto4.0 用户手册”