**从树莓派4构建理解buildroot原理**

**1 buildroot 背景知识**

总结一下：

1. buildroot提供一种soc嵌入式的包管理工具（类似ubuntu apt-get包管理器，python的pip包管理器），解决不同库的版本以及依赖问题，但注意不同buildroot支持的库多少以及版本有差异（比如edge10 buildroot版本较低仅支持2000+包，而最新的buildroot支持3000+包），尽量选择高版本的buildroot；
2. buildroot提供缺省的交叉工具链构建、根文件系统构建以及镜像制作相关的配套工具；
3. buildroot提供的各种包集成方式（包括包下载、补丁、编译构建、安装等），对于自己手动安装的包，请添加各个包的依赖关系，比如hcp库依赖hal,mal，需要增加dependency。这样如果修改hal，那么会自动rebuild hcp；
4. Linux kernel也属于package，不是必须构建项。绝大部分包以及用户包只会依赖工具链的glibc，不会依赖具体的kernel，但打包镜像一般会依赖kernel。
5. 如果package是基于开源做少量定制开发的，比如Linux kernel，opensbi，uboot等，厂商一般通过patch方式发布；
6. Buildroot 是基于make系统构建的，支持任意中间target的构建，也支持单独的镜像pack打包（伪目标）；

**1.1 介绍**

Buildroot是一个开源工具，用于帮助开发者为嵌入式系统生成交叉编译工具链、根文件系统、内核映像以及可引导的固件映像。由于嵌入式系统的资源有限，并且需要特定的配置和优化，因此Buildroot为开发者提供了一个简化整个构建过程的框架。

关于Buildroot的一些核心特点和功能：

1. 配置界面：Buildroot使用menuconfig风格的配置界面（与Linux内核配置类似），允许开发者选择所需的软件包、库和配置选项。
2. 交叉编译工具链：Buildroot可以为目标架构生成专用的交叉编译工具链。开发者可以选择使用预编译的工具链，或者完全从源代码构建。
3. 软件包系统：Buildroot包含数千个预配置的软件包，如常见的库、工具和应用程序。这些软件包可以轻松地添加到最终的根文件系统映像中。
4. 文件系统映像：Buildroot支持生成多种文件系统格式的映像，如ext2/3/4、squashfs、ubifs等，这些映像可以直接烧录到目标设备。
5. 引导加载程序和内核：除了根文件系统，Buildroot还可以构建引导加载程序（如U-Boot、GRUB）和Linux内核映像。
6. 定制化和扩展性：开发者可以为Buildroot添加自定义的软件包和补丁，以满足特定的需求。这使得Buildroot非常灵活，并能够支持各种不同的嵌入式项目。
7. 构建自动化：Buildroot自动化了整个构建过程，从下载源代码、应用补丁、编译软件包到生成最终的固件映像，这确保了构建的一致性和可重复性。
8. 目标优化：Buildroot提供了多种优化选项，以减小最终映像的大小，这对于资源有限的嵌入式设备尤为重要。

**1.2 构建步骤**

1. 配置：  
   用户通过make menuconfig或其他类似的命令选择所需的软件包、版本、库和配置选项。这些配置被保存在.config文件中。
2. 工具链构建：  
   根据用户的选择，Buildroot可能会从源代码开始构建交叉编译工具链，或者使用预编译的交叉编译工具链。
3. 软件包下载：  
   Buildroot为所选的每个软件包下载相应的源代码。它使用每个软件包的.mk文件中指定的URL和哈希值来下载。
4. 软件包构建：  
   一旦所有源代码都已下载，Buildroot会按照依赖关系的顺序开始编译软件包。每个软件包的构建过程都包括配置、编译和安装步骤。
5. 根文件系统创建：  
   所有软件包都被编译并安装到一个临时的目标目录中。然后，Buildroot从这个目录中生成所选格式的根文件系统映像，如ext4、squashfs等。
6. 引导加载程序和内核：  
   如果用户选择了，Buildroot还会构建引导加载程序和Linux内核。这些组件与根文件系统一起被烧录到目标设备。
7. 最终映像：  
   在某些情况下，Buildroot可以生成一个完整的二进制映像，其中包括引导加载程序、内核、设备树和根文件系统，这可以直接烧录到SD卡或其他存储介质上。

**1.3 Makefile**

Buildroot主要使用Makefile来编写其构建规则。这些Makefile定义了如何下载、配置、编译和安装各个软件包，以及如何生成交叉编译工具链、内核映像和根文件系统映像。

以下是Buildroot中Makefile使用的一些关键点：

1. 顶层Makefile：Buildroot的主目录中有一个顶层Makefile，它为整个构建过程提供了入口点。
2. 软件包Makefile：对于Buildroot中的每个软件包，都有一个与之相关的.mk文件。这个文件包含了关于如何下载、解压、配置、编译和安装该软件包的规则。这些.mk文件通常位于package/目录下的子目录中。
3. 配置框架：Buildroot使用Kconfig系统来管理配置选项，这也是Linux内核使用的配置系统。尽管这不是Makefile本身的一部分，但Makefile会引用这些配置来决定哪些软件包应该被构建、哪些选项应该被启用等。
4. 函数和宏：Buildroot定义了许多Makefile函数和宏，以简化和标准化常见的任务，如下载源代码、应用补丁或构建自动工具。
5. 钩子和覆盖：Buildroot提供了机制，允许用户为特定的软件包或阶段提供自定义的Makefile钩子或覆盖，从而提供更高的定制性。

**1.4 package包**

在Buildroot中，软件包（package）的构建主要依赖于各个软件包自身的构建系统。Buildroot支持多种常见的构建系统，并为这些构建系统提供了通用的接口和规则，使得整个构建过程更为统一和自动化。以下是Buildroot支持的主要软件包构建方式：

1. Autotools：  
   这是一个由autoconf和automake工具组成的构建系统，用于自动生成configure脚本和Makefile。Buildroot为Autotools提供了标准的./configure && make && make install构建步骤。
2. CMake：  
   CMake是一个跨平台的构建系统，它生成与平台无关的构建文件。Buildroot支持通过CMake生成Makefile或其他构建文件，并使用它们来构建软件包。
3. Meson：  
   Meson是一个较新的构建系统，使用Ninja作为其后端。Buildroot为Meson提供了规则，使其能够正常工作。
4. Plain Makefile：  
   有些软件包使用纯粹的Makefile（无configure脚本或其他自动生成工具）。Buildroot直接调用make来构建这些软件包。
5. Python setuptools：  
   对于Python软件包，Buildroot支持使用setuptools进行构建和安装。
6. Qmake：  
   Qmake是Qt项目的构建工具。Buildroot支持使用Qmake构建和安装Qt应用程序和库。
7. Custom Build System：  
   对于不使用上述任何标准构建系统的软件包，Buildroot允许开发者提供自定义的构建和安装命令。

除了直接的构建步骤，Buildroot还处理其他与软件包构建相关的任务，如：

* 下载：从指定的URL下载软件包的源代码。
* 解压：解压下载的源代码归档文件。
* 补丁：应用为特定软件包提供的补丁。
* 安装：将构建的软件包安装到一个目标目录，从中生成根文件系统映像。

总的来说，Buildroot为各种常见的构建系统提供了统一的接口和规则，使得为嵌入式系统构建软件包变得更为简单和自动化。

输出介绍

**1.5 输出**

|  |  |  |
| --- | --- | --- |
| 阶段 | 输出物 | 简介 |
| build阶段 | .config | Buildroot的主配置文件,决定了哪些软件包会被构建、它们的配置选项以及其他与构建过程相关的设置. |
| Makefile | Top makfile，调用buildroot的Makefile。 |
| build/ | 目录包含了所有软件包的构建产物。对于每个软件包，build/目录下都有一个与软件包名称相对应的子目录。这些子目录中包含了从源代码到二进制文件的构建过程中产生的所有文件。 |
| install阶段 | host/ | Buildroot构建的所有主机工具和库的安装目录。这包括交叉编译工具链、用于包管理的工具、库以及其他为构建目标系统而在主机上需要的工具。 |
| staging/ | 目标系统的“预安装”目录，其中包含了所有为目标系统构建的库和头文件。它的结构与最终的根文件系统非常相似，但主要用于编译其他软件包，以确保它们链接到正确的库和使用正确的头文件 |
| target/ | 目标根文件系统（root filesystem）的内容。target/目录包含了所有你选择安装的软件包、库、二进制文件、配置文件、脚本等，这些内容都将出现在最终的设备上 |
| images/ | 目录是构建过程的最终输出目录。它包含了为目标硬件平台构建的所有必要文件和映像，这些文件和映像将被部署到设备上。通常包括Linux内核、根文件系统映像、引导加载程序以及其他相关的二进制文件。 |

我们可以参考package的mk文件。一般包在build目录下构建，然后根据package的mk写法install到host/, target/ staging/，images/。

1. PACKAGE\_INSTALL\_HOST\_CMDS：指定如何将包的内容安装到host/目录。
2. PACKAGE\_INSTALL\_STAGING\_CMDS：指定如何将包的内容安装到staging/目录。如果一个包提供库或头文件，那么这些库和头文件会被安装到staging/目录，以便其他包在编译时可以链接到这些库并使用这些头文件。这确保了所有软件包都使用了正确和一致的库版本。
3. PACKAGE\_INSTALL\_TARGET\_CMDS：指定如何将包的内容安装到target/目录。如果一个软件包提供了在目标系统上运行所需的二进制文件、库、脚本或配置文件，那么它应该被安装到target/目录。target可以理解为运行时需要的包，如果单纯的开发包（中间结果,只为其他软件包的构建提供头文件和库）是不需要放到target目录的。
4. PACKAGE\_INSTALL\_IMAGES\_CMDS：指定如何将包的内容安装到image目录。

**2 树莓派的buildroot构建**

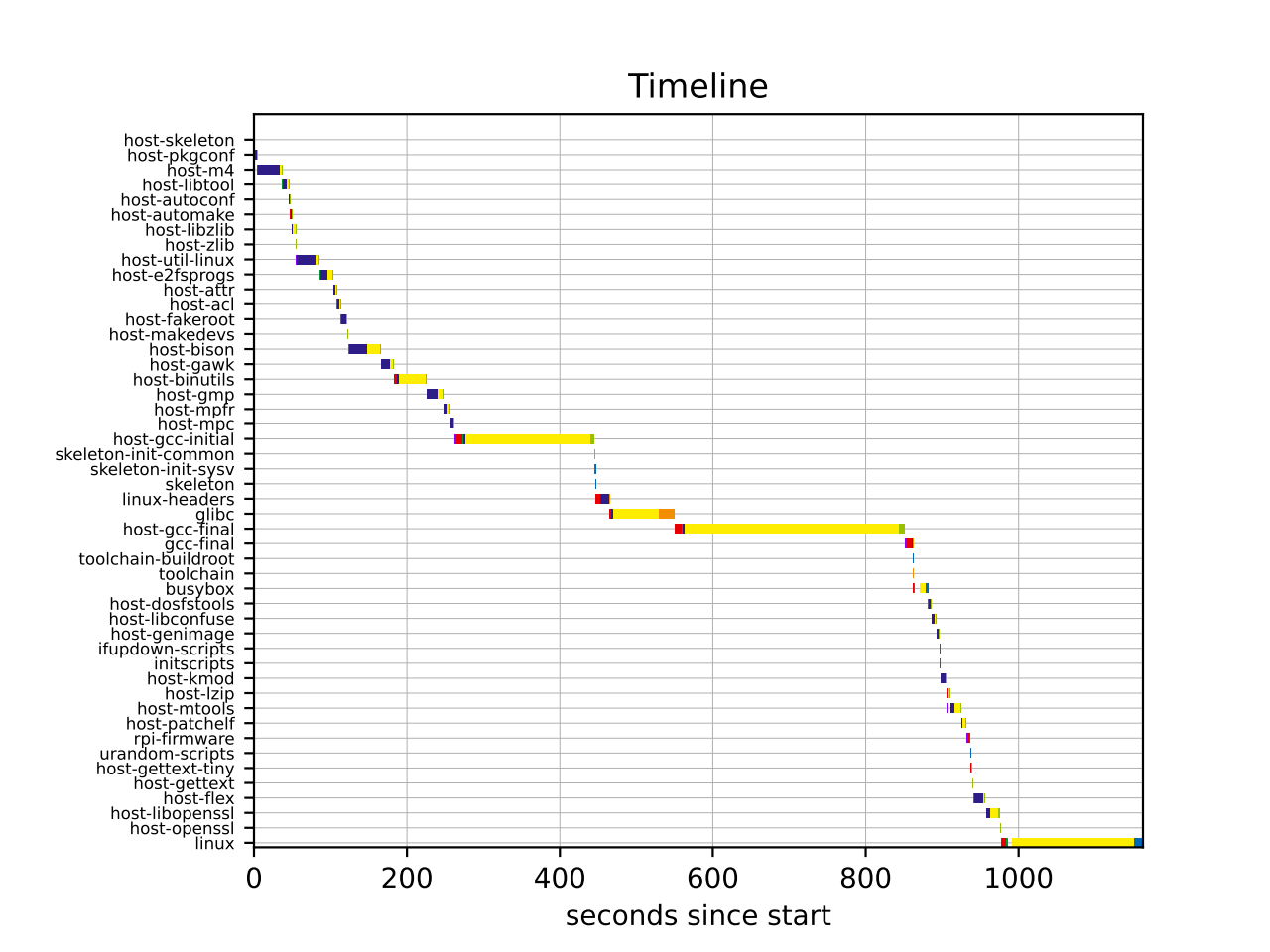
**2.1 树莓派的构建方法**

1. 下载buildroot： https://github.com/buildroot/buildroot
2. make list-defconfigs ：列出官方支持的配置文件
3. make O=output/raspberrypi4/ raspberrypi\_defconfig， 选择树莓派4配置，输出为output/raspberrypi4/.config文件。
4. make O=output/raspberrypi4/ -j32 编译all；
5. make O=output/raspberrypi4/ pkt-target ： 编译指定的pkt-target
6. 工具：

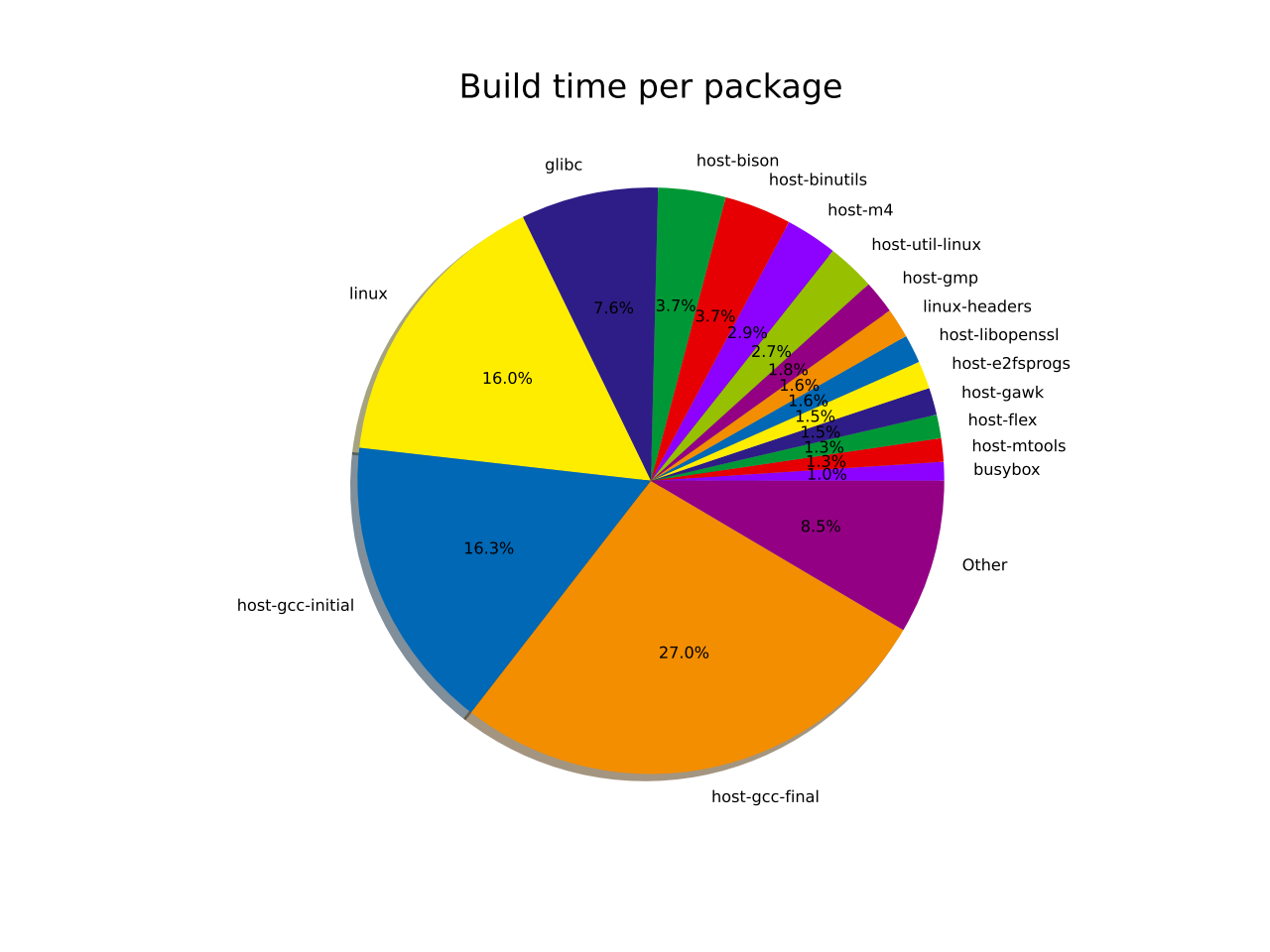
* make O=output/raspberrypi4/ graph-depends：各个targets的依赖关系；
* make O=output/raspberrypi4/ graph-build：编译的perf统计信息；

**make最终输出为images/sdcard.img.**

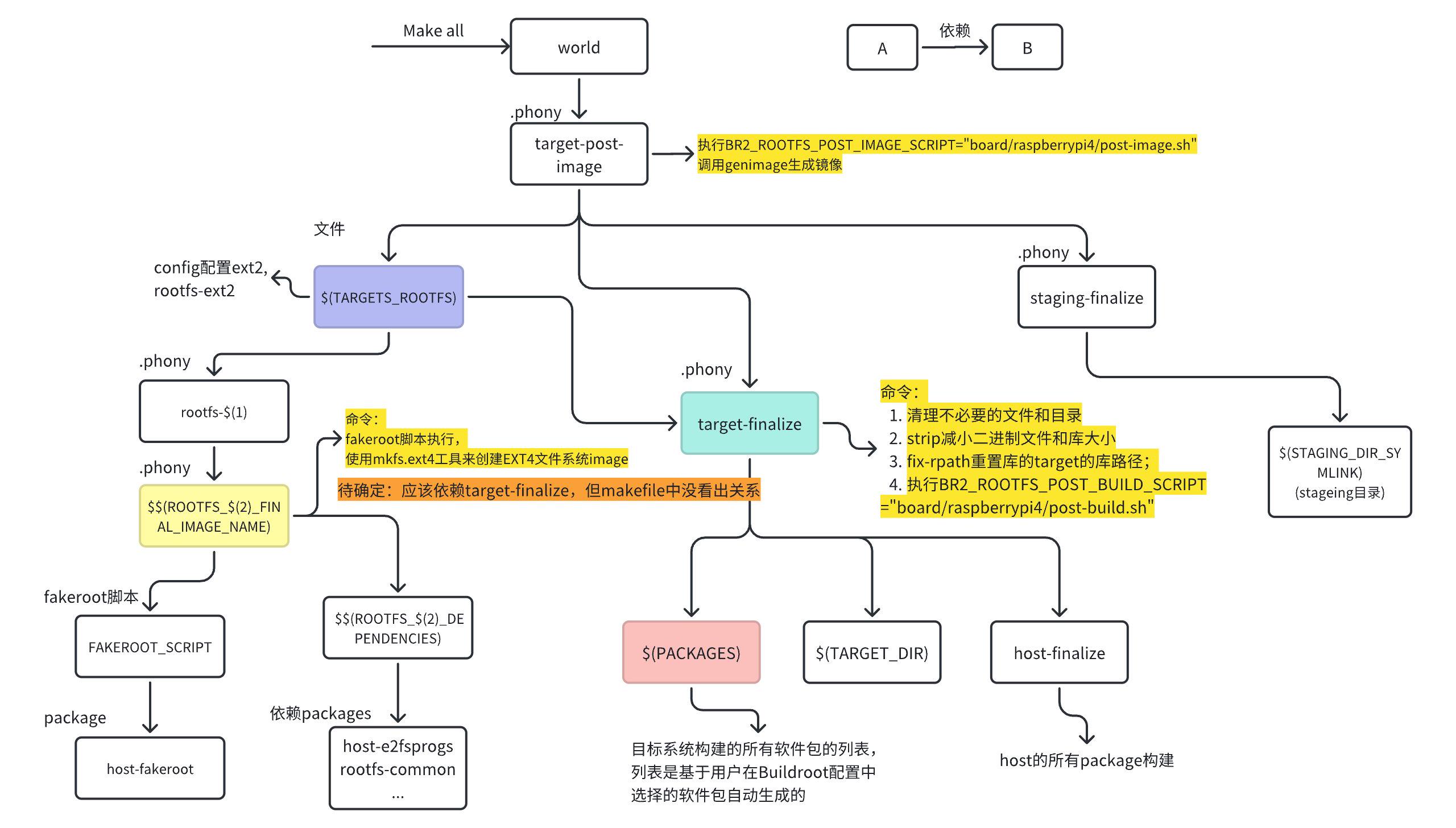
graph-build的build.timeline: 可以看到各个package的构建顺序以及时间开销。



graph-build的build.pie-packages：可以看到各个package的build时间。



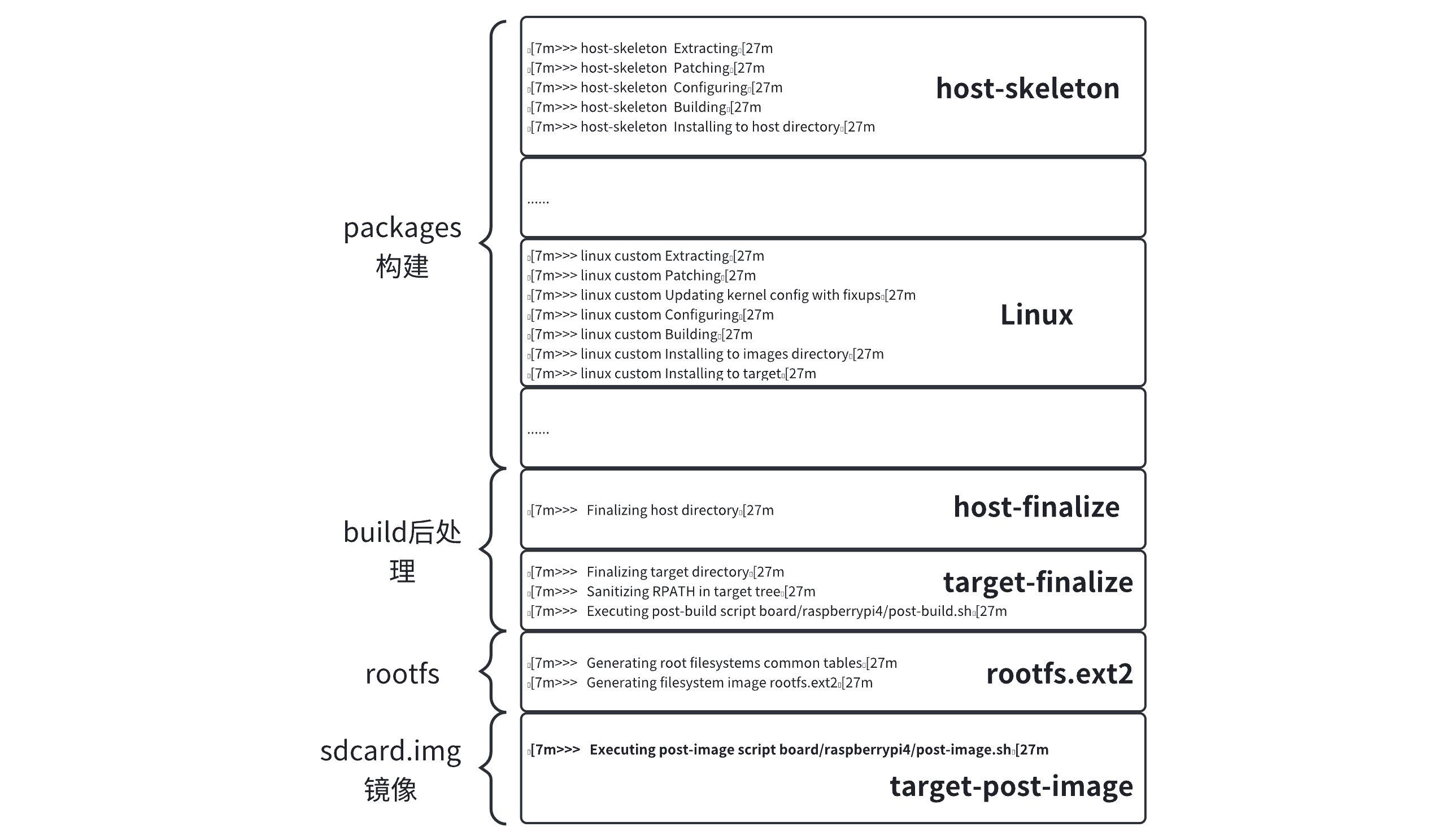
**2.2 Makefile分析**



packages信息参考output的：

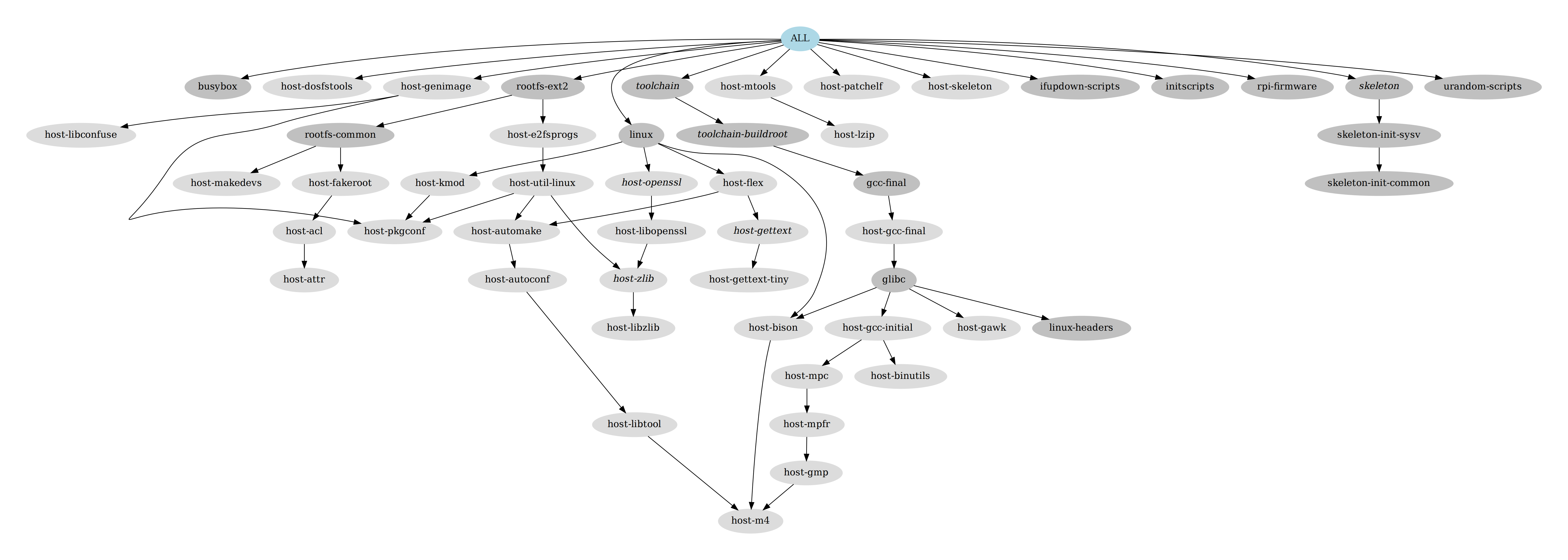
1. packages-file-list-host.txt
2. packages-file-list-staging.txt
3. packages-file-list.txt

结合树莓派builds log来分析：



**2.3 树莓派的构建分析**

**2.3.1 包类型**



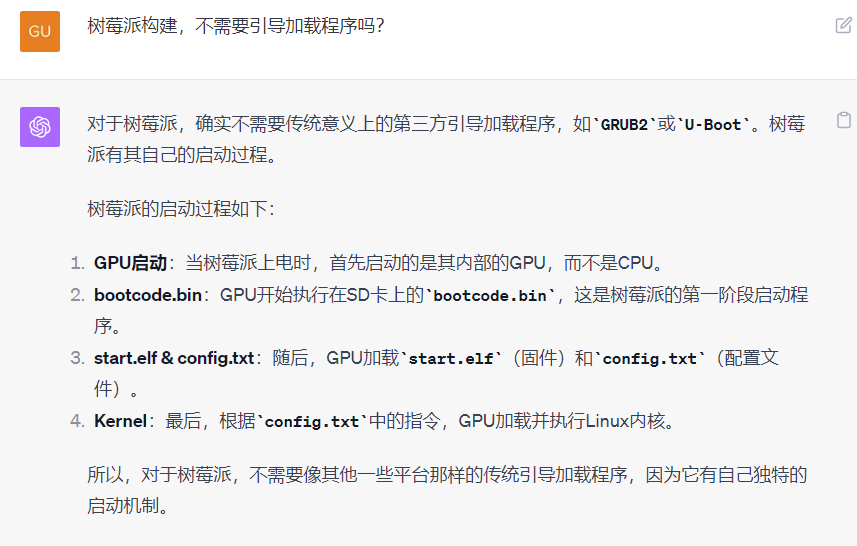
上图build的package依赖关系，从中可以看到两类package：

1. host-xxx：在host主机上构建的包，这些包大部分属于工具类，辅助目标机的包构建以及镜像打包等；
2. xxx：在目标机上构建的包，主要是busybox，linux, rootfs（三方库）, rpi-firmware等；



**2.3.2 boot配置**

raspberrypi\_defconfig中没有启动任何boot设置。



**2.3.3 linux kernel（custom定制）**

https://github.com/raspberrypi/linux

采用linux tar包方式，内部包括：

1. 内核配置文件： "bcm2711"；
2. DTS文件：bcm2711-rpi-4-b

**2.3.4 rpi-firmware**

https://github.com/raspberrypi/firmware

https://github.com/buildroot/buildroot/blob/29b6e170b7337c47444a8ef34529eabc2c59567f/package/rpi-firmware/rpi-firmware.mk#L4

树莓派的firmware是采用prebuild预编译方式，直接install到buildroot的output目录里。

**2.3.5 raspi-gpio(可选）**

https://github.com/RPi-Distro/raspi-gpio

https://github.com/buildroot/buildroot/blob/29b6e170b7337c47444a8ef34529eabc2c59567f/package/raspi-gpio/raspi-gpio.mk#L4

树莓派gpio package是采用autotools构建的。$(eval $(autotools-package)

**2.3.5 构建输出**

参见https://github.com/buildroot/buildroot/blob/79e01ef9506a6cdc4836912607dc594ae7b1999d/board/raspberrypi/post-image.sh#L32

genimage的配置参数如下：

|  |
| --- |
| Plain Text image boot.vfat {  vfat {  files = {  "bcm2711-rpi-4-b.dtb",  "rpi-firmware/cmdline.txt",  "rpi-firmware/config.txt",  "rpi-firmware/fixup4.dat",  "rpi-firmware/overlays",  "rpi-firmware/start4.elf",  "zImage",   }  }   size = 32M }  image sdcard.img {  hdimage {  }   partition boot {  partition-type = 0xC  bootable = "true"  image = "boot.vfat"  }   partition rootfs {  partition-type = 0x83  image = "rootfs.ext4"  } } |

配置定义了两个映像：boot.vfat 和 sdcard.img。

1. boot.vfat:

* 这是一个VFAT文件系统映像。
* 它包含了一系列文件，如bcm2711-rpi-4-b.dtb（设备树二进制文件）、rpi-firmware/下的启动文件、zImage（Linux内核映像）等。
* 这个映像的大小被设置为32MB。

1. sdcard.img:

* 这是一个硬盘映像，代表整个SD卡。
* 它有两个分区：boot 和 rootfs。
* boot 分区：
* 类型为0xC，这是一个FAT32分区。
* 它是可引导的。
* 它使用了先前定义的boot.vfat映像。
* rootfs 分区：
* 类型为0x83，这是一个Linux文件系统分区。
* 它使用了一个名为rootfs.ext4的映像（可能是由mkfs.ext4创建的）。