

Tutorial-HPS-Buildroot



Buildroot wikipedia

Buildroot is a set of Makefiles and patches that simplifies and automates the process of building a complete and bootable Linux environment for an embedded system, while using cross-compilation to allow building for multiple target platforms on a single Linux-based development system. Buildroot can automatically build the required cross-compilation toolchain, create a root file system, compile a Linux kernel image, and generate a boot loader for the targeted embedded system, or it can perform any independent combination of these steps. For example, an already installed cross-compilation toolchain can be used independently, while Buildroot only creates the root file system

- ref: <https://en.wikipedia.org/wiki/Buildroot>



Iremos utilizar o **buildroot** para gerar o filesystem (`/bin` , `/etc` , ...) do nosso sistema embarcado. No buildroot teremos a opção de configurar quais softwares queremos no dispositivo. Por exemplo, se desejarmos acessar o HPS via ssh, teremos que no buildroot adicionar um ssh server para ser compilado e adicionado ao filesystem e executado no boot.

O buildroot é uma alternativa a outro projeto bem conhecido: Yocto. O vídeo a seguir são dois desenvolvedores, um de cada projeto, fazendo a comparação entre as duas ferramentas:



Note

O Yocto está se consolidando como ferramenta padrão da indústria, tomando o lugar do buildroot. A escolha pelo buildroot na eletiva é pela facilidade de criar um sistema, o yocto é mais complexo e cheio de terminologias. Nessa eletiva iremos trabalhar com o buildroot, mas para quem quer se aprofundar/especializar no tema, tem que aprender o yocto.



buildroot



Leitura recomendada

- https://buildroot.org/downloads/manual/manual.html#_getting_started

Download

Primeiramente devemos fazer o download do `buildroot` :

```
$ git clone https://github.com/buildroot/buildroot
$ cd buildroot/
```

O buildroot possui uma ferramenta de configuração similar ao do kernel do linux (`menuconfig` / `nconfig`) iremos utilizar-la para configurar o filesystem assim como quais programas serão compilados e inseridos no `/root/` . Lembre que já possuímos um toolchain (o que compilamos o kernel) configurado no `.bashrc` , iremos o utilizar para a compilação de todos os programas que iremos carregar no embarcado.

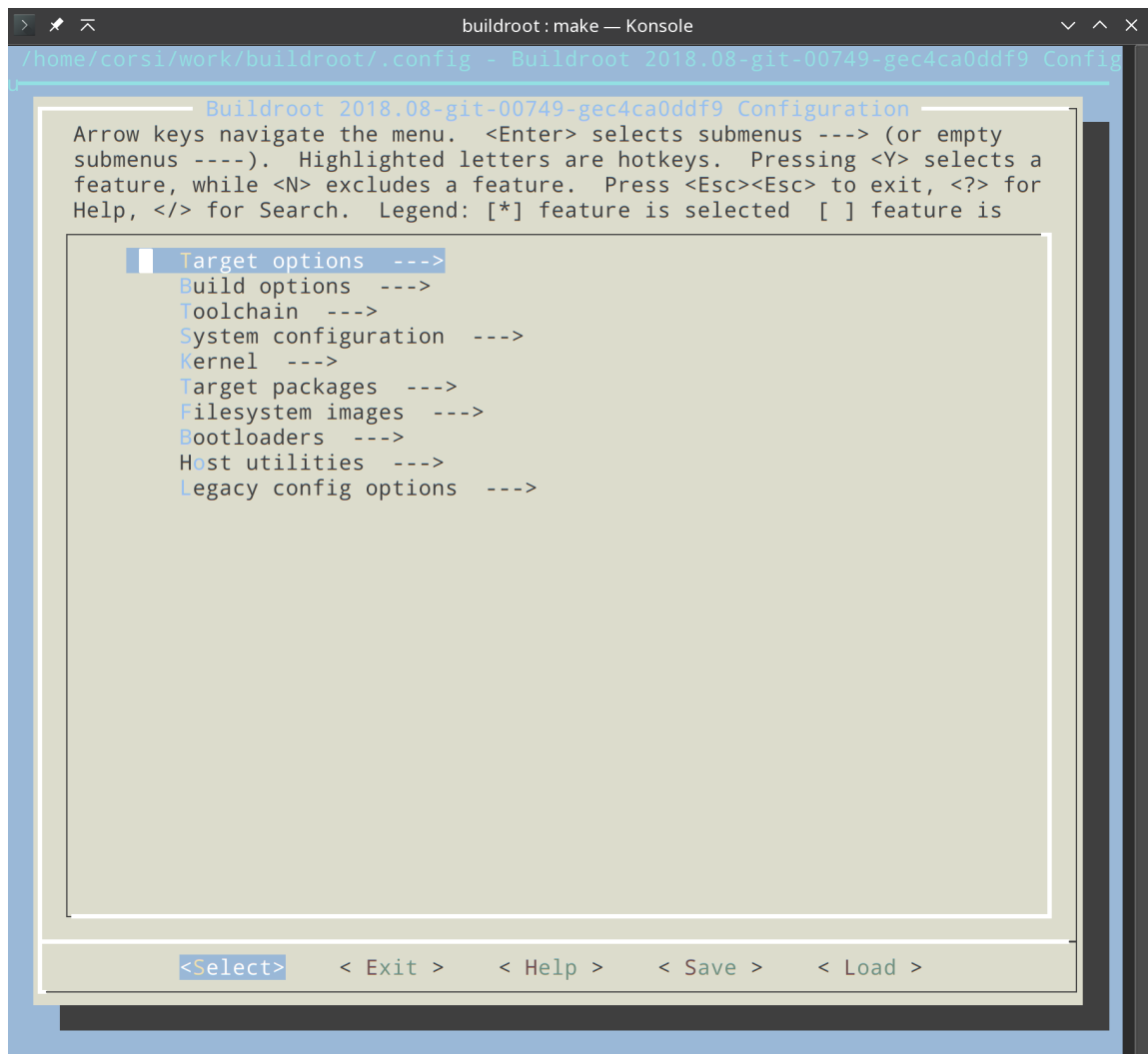
O buildroot tem a opção de fazer o download do toolchain (ele também pode compilar o kernel e gerar o uboot, é uma ferramenta bem completa), mas dessa vez iremos utilizar o que já temos (para manter a compatibilidade).

Configurando

Na pasta do buildroot recém clonada, execute o seguinte comando:

```
$ make ARCH=arm menuconfig
```

Ele irá abrir uma tela de configuração a seguir:



para voltar para essa tela, basta aperta duas vezes a tecla <ESC>

1. Target Options

A primeira parte que iremos configurar é o alvo da geração do filesystem (*Target options*), devemos informar para o buildroot que ele está gerando arquivos para um ARM e indicar algumas opções do nosso compilador. Para isso:




```

1: make ARCH=arm menuconfig
/home/corsi/work/buildroot/.config - Buildroot 2019.11-git-00335-g9ab72c73cc Configu
r> Target options
                                Target options
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y> selects a
feature, while <N> excludes a feature. Press <Esc><Esc> to exit, <?> for
Help, </> for Search. Legend: [*] feature is selected [ ] feature is

[*] Target Architecture (ARM (little endian)) --->
Target Binary Format (ELF) --->
Target Architecture Variant (cortex-A9) --->
[*] Enable NEON SIMD extension support
[*] Enable VFP extension support
Target ABI (EABIhf) --->
Floating point strategy (NEON) --->
ARM instruction set (ARM) --->

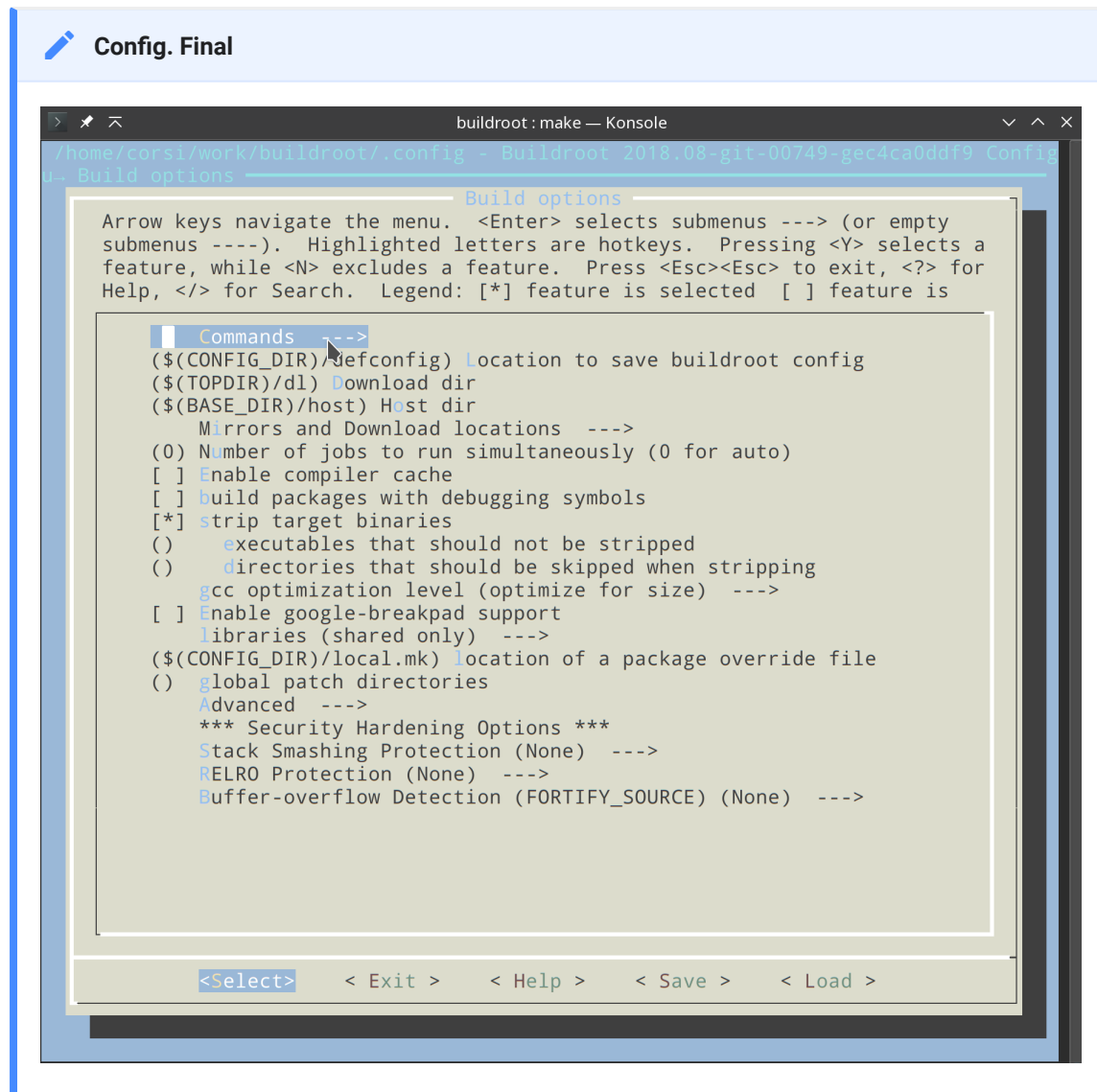
<Select>  <Exit >  <Help >  <Save >  <Load >

```

- Menu principal  Target Options
 - Target Architecture: **ARM (little endian)**
 - Essa opção já deve estar certa pois passamos via a chamada do make (make ARCH=ARM ...)
 - Target Architecture Variant: **cortex-A9**
 - **Enable** NEON SIMD extension support
 - **Enable** VFP extension support
 - Floating point strategy: **NEON**
 - <https://developer.arm.com/technologies/neon>

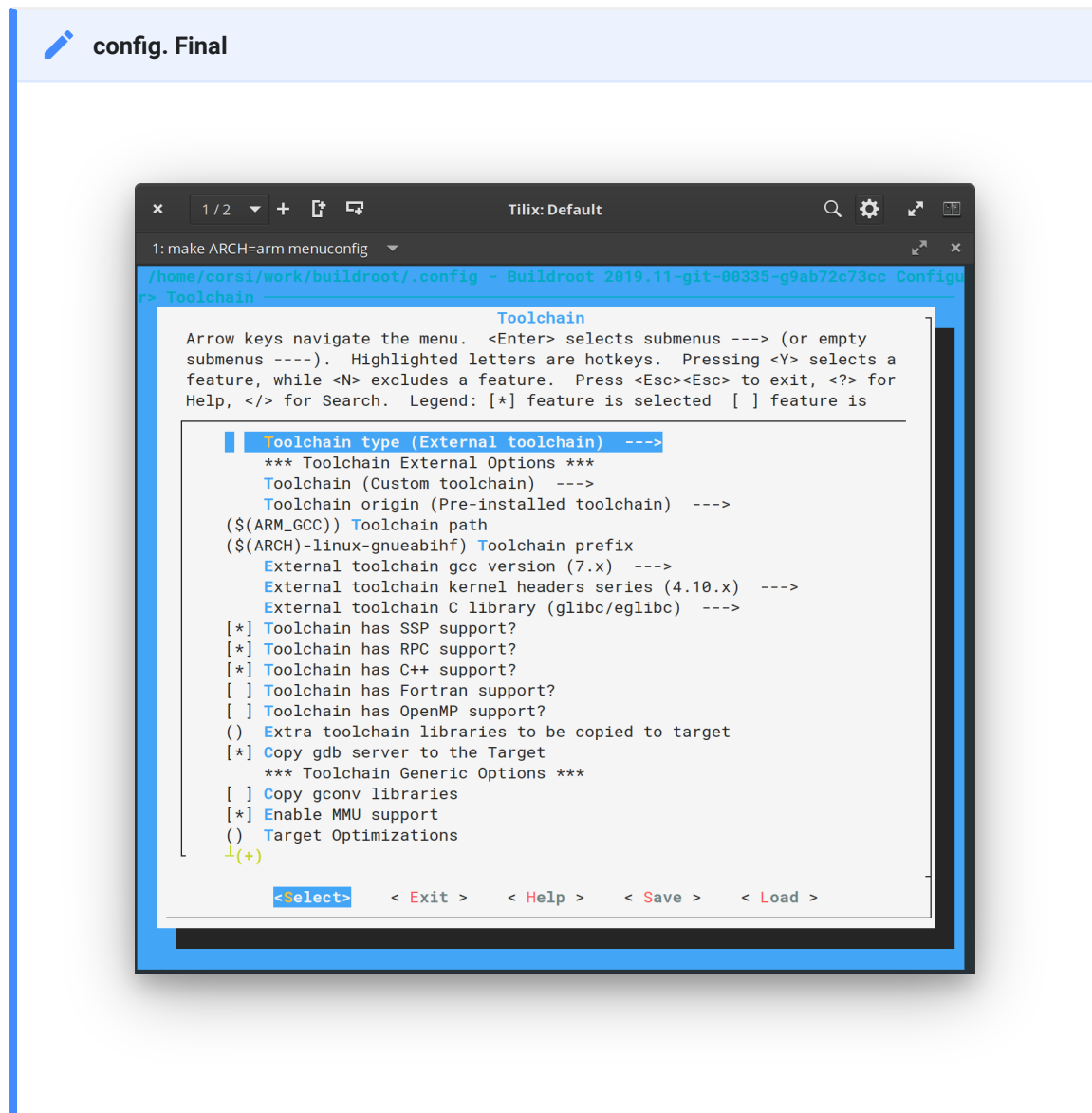
- Target ABI: **EABIhf**
 - Indicamos ao buildroot que nossa arquitetura possui ponto flutuante em HW.

2. Build options



Deixe padrão como o padrão.

3. Toolchain



Vamos indicar agora para o buildroot qual toolchain que ele deve utilizar e suas configurações:

- Menu principal → Toolchain
 - Toolchain type: **External toolchain**
 - o buildroot irá usar o toolchain que especificarmos. Note que dentro dessa opção existe a : *Buildroot toolchain*, que se ativada faria com que o buildroot baixasse de forma automática todo o toolchain.
 - Toolchain: **Custom toolchain**

- Toolchain path: **`$(ARM_GCC)`**
 - o buildroot irá usar essa variável do sistema como path do toolchain. Temos duas opções aqui :
 - a. Podemos declarar essa variável no bash
 - b. Podemos editar essa opção já com o path do nosso toolchain
 - c. Vamos escolher por hora a opção 1.
 - Toolchain prefix: **`$(ARCH)-linux-gnueabi`**
 - o prefix é como o toolchain irá ser chamado, por exemplo para acessar o gcc:
 - `$(ARM_GCC)/bin/$(ARCH)-linux-gnueabi-gcc`
 - Sendo :
 - `ARM_GCC = /home/corsi/work/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi`
 - `ARCH = arm` (passado no call do make)
 - Resulta em: `/home/corsi/work/gcc-linaro-7.1-2017.11-x86_64_arm-linux-gnueabi/bin/arm-linux-gnueabi-gcc`
 - Toolchain gcc version: **7.x**
 - Toolchain kernel headers series: **4.10.x**
 - External toolchain C library: **glibc/eglibc**
 - **Ativar:** Toolchain has SSP support
 - **Ativar:** Toolchain has SSP support
 - **Ativar:** Toolchain has RCP support
 - **Ativar:** Toolchain has C++ support
-

4. System Configuration



config. Final

```
1: make ARCH=arm menuconfig
/home/corsi/work/buildroot/.config ~ Buildroot 2019.11-git-00335-g9ab72c73cc Configu
r> System configuration

System configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y> selects a
feature, while <N> excludes a feature. Press <Esc><Esc> to exit, <?> for
Help, </> for Search. Legend: [*] feature is selected [ ] feature is

Root FS skeleton (default target skeleton) --->
(de10Standard) System hostname
(Computacao Embarcada!! SoC Cyclone V) System banner
[*] Passwords encoding (sha-256) --->
Init system (BusyBox) --->
/dev management (Dynamic using devtmpfs only) --->
(system/device_table.txt) Path to the permission tables
[ ] support extended attributes in device tables
[ ] Use symlinks to /usr for /bin, /sbin and /lib
[ ] Enable root login with password
/bin/sh (busybox' default shell) --->
[*] Run a getty (login prompt) after boot --->
[*] remount root filesystem read-write during boot
() Network interface to configure through DHCP
(/bin:/sbin:/usr/bin:/usr/sbin) Set the system's default PATH
[*] Purge unwanted locales
(C en_US) Locales to keep
() Generate locale data
[ ] Enable Native Language Support (NLS)
[ ] Install timezone info
↑(+)
```

<Select> <Exit> <Help> <Save> <Load>


Nessa etapa vamos configurar informações como: hostname, user, password gerenciador de inicialização (init)...

- Menu principal → System Configuration
 - System hostname: **SoC-Corsi** (escolha o que preferir)
 - System banner: **Embarcados Avancados!! SoC Cyclone V**
 - Init system: **BusyBox**
 - systemd é uma alternativa, só que mais complexa!
 - Root password: **1234** (escolha o que preferir)

- /bin/sh: **busybox**
 - O shell a ser inserido no sistema, temos várias outras opções: bash, zsh. Todas elas irão aumentar o tamanho e a complexidade da imagem.
-

5. Kernel / bootloader

O busybox pode baixar e compilar o kernel e o uboot para nós. Não vamos usar essa configuração.

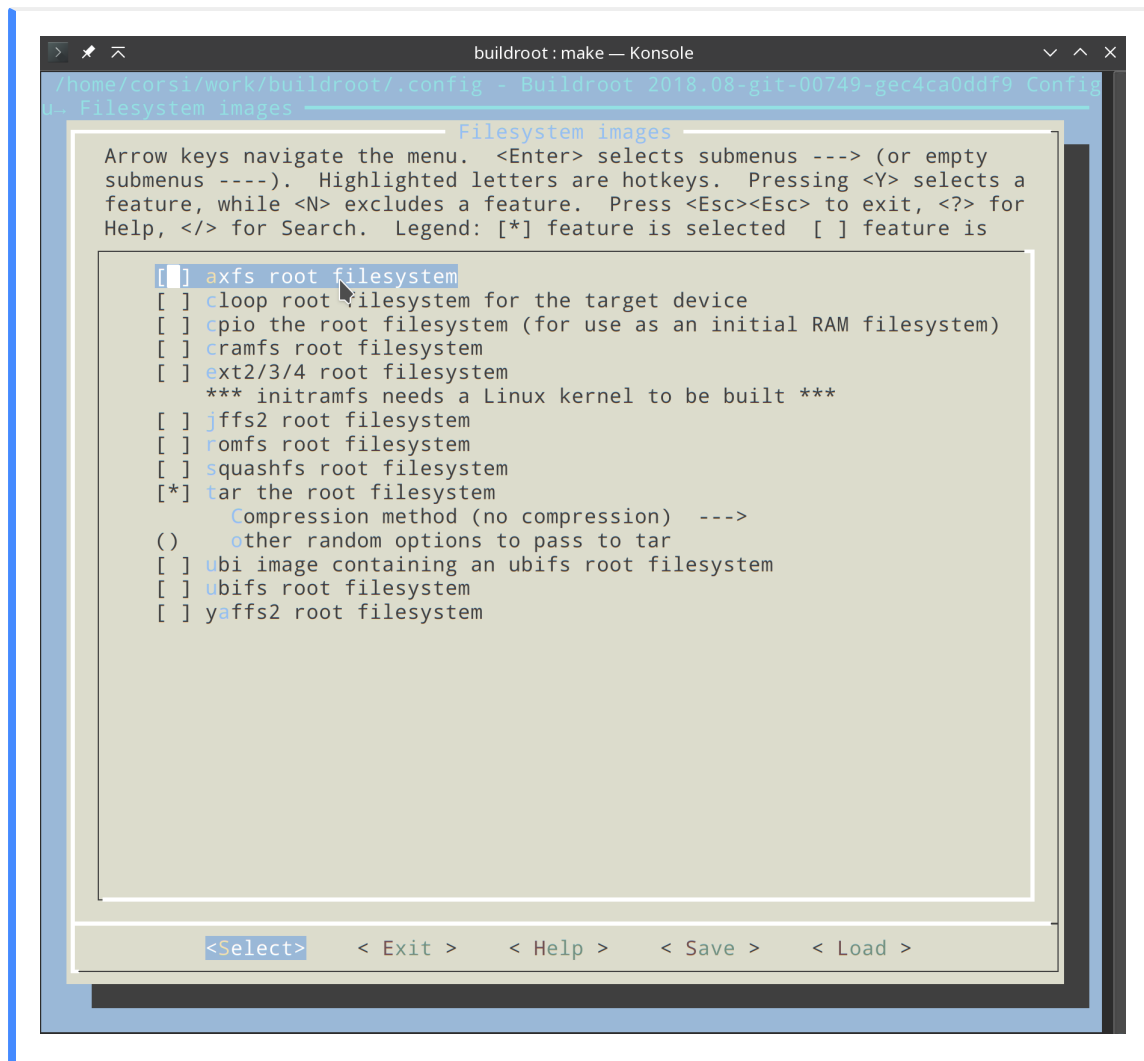
- Menu principal  Kernel
-

6. Target packages

Nesse menu temos a opção de quais programas e sistemas serão inseridos na imagem para o target. Se quisermos por exemplo inserir um webserver (apache ?) no nosso linux embarcado, devemos selecionar aqui.

Vamos deixar como padrão por hora. Mais tarde iremos voltar a essa etapa mais tarde.

7. Filesystem images



- Menu principal ➡ Filesystem images
 - Selecionar: **tar the root filesystem**

Esse menu descreve para o busybox como deve ser a saída final da imagem do filesystem gerada. O busybox necessita gerar filesystem que é capaz de configurar as permissões dos arquivos corretamente (ele não pode simplesmente gerar uma pasta com todos os arquivos e programas).

8. Finalizando

Salve a sua configuração (ESC ESC save) e volte ao terminal. Vamos agora gerar a imagem do nosso filesystem.

Compilando

Para compilar e gerar o filesystem :

```
$ make all -j 4
```

Nessa etapa o buildroot irá baixar da web todos os pacotes e programas que foram selecionados no menu de configuração, e irá compilar o source code com o toolchain que passamos para ele. **Isso pode levar um tempinho.**

Gráficos !

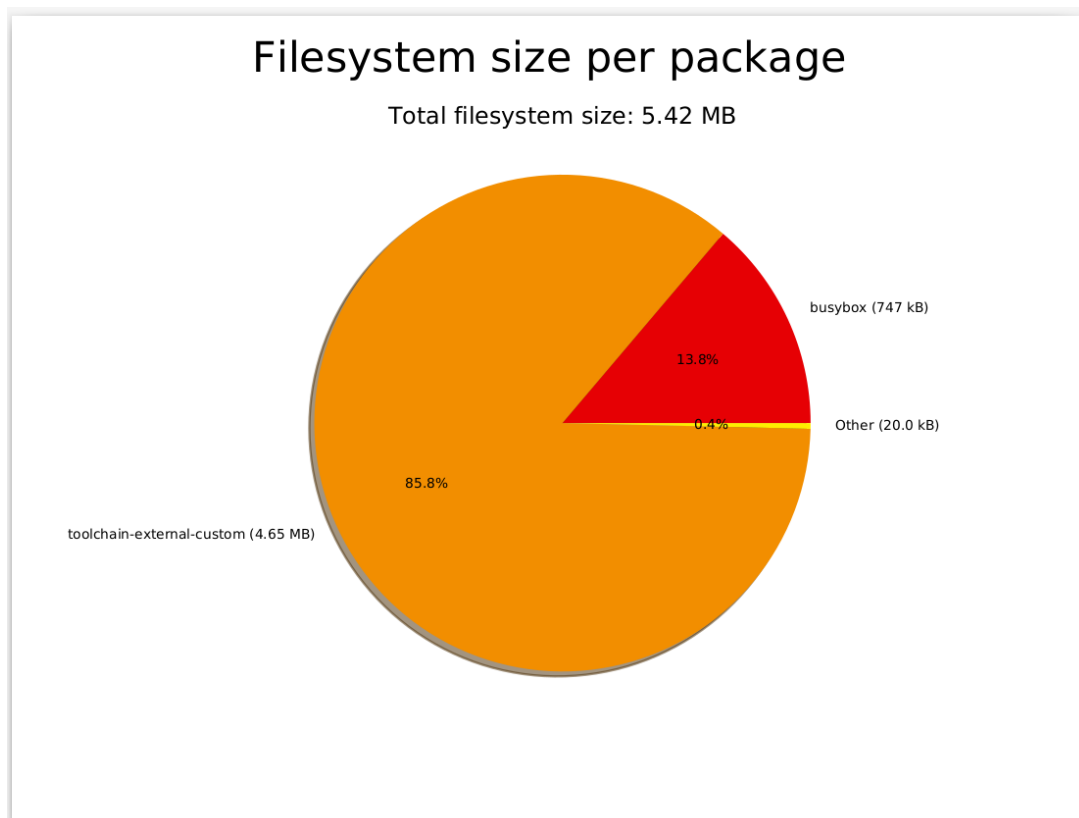
Uma vez acabado o processo de geração do FS, podemos gerar alguns gráficos muito importantes:

- https://buildroot.org/downloads/manual/manual.html#_graphing_the_filesystem_size_contribution_of_packages
- Dependência dos pacotes: `make graph-depends`
- Tempo de compilação: `make graph-build`
- Contribuição do tamanho do FS de cada pacote: `make graph-size`

Os gráficos são salvos na pasta: `output/graphs/`

Gere os três gráficos e analise os resultados

Exemplo do gráfico do tamanho dos pacotes no fs:



Outputs

Existem duas saídas do buildroot na pasta: `buildroot/output/**`

1. O arquivo `./images/rootfs.tar` : que contém o fileSystem do target (com as permissões corretas)
2. A pasta `./images/target/` : com os arquivos contidos no `.tar` mas sem as permissões corretas para executar no target. Inclusive essa pasta possui um arquivo:

Warning

THIS_IS_NOT_YOUR_ROOT_FILESYSTEM

Warning!

This directory does *not* contain the root filesystem that you can use on your embedded system. Since Buildroot does not run as root, it cannot create device files and set the permissions and ownership of files correctly in this directory to make it usable as a root filesystem.

Para testarmos no nosso sistema embarcados, temos que extrair o arquivo `rootfs.tar` para o nosso cartão de memória.

Testando

Siga o tutorial em [SDCard - FileSystem](#). Lá está comentando como extrair o `rootfs.tar` para o nosso cartão de memória.

1. o boot ficou mais rápido?
2. Tente plugar um pendrive, funciona?

Estudando

Responda:

- descreva o que é o root file system
- initd process
- para que serve e como funciona o `/linuxrc`
- para que serve o `/proc`

Referências

- dtb : [https://rocketboards.org/foswiki/Documentation/](https://rocketboards.org/foswiki/Documentation/HOWTOCreateADeviceTree)
[HOWTOCreateADeviceTree](#)

- Generating and Compiling the Preloader : <https://rocketboards.org/foswiki/Documentation/GSRD141Preloader>
- Compilando o kernel : <https://rocketboards.org/foswiki/Documentation/EmbeddedLinuxBeginnerSGuide#8>