LABORATÓRIO TPSE I

Laboratório 05: Uso Básico do Buildroot

Prof. Francisco Helder

29 de setembro de 2022

O objetivo deste laboratório é criaremos um sistema Linux completo com o Buildroot e depois montaremos este sistema na placa específica. Veremos como essa ferramenta facilita bastante o processo de desenvolvimento em Linux embarcado. Após este laboratório, você será capaz de:

- Obter Buildroot
- Configure um sistema mínimo com Buildroot para o BeagleBone Black
- Faça a construção
- Prepare o BeagleBone Black para uso
- Gravar e teste o sistema gerado

1 Baixando o Buildroot

Já que vamos fazer o desenvolvimento do Buildroot, vamos clonar o código-fonte do Buildroot de seu repositório Git:

```
$ git clone https://git.buildroot.net/buildroot
```

Crie um diretório buildroot e entre nele. Vamos iniciar um branch a partir da versão 2022.08 do Buildroot.

```
$ git checkout -b bootlin 2022.08
```

2 Configurando o Buildroot

Se você olhar em **configs/**, verá que existe um arquivo chamado beaglebone_defconfig, que é um arquivo de configuração Buildroot pronto para usar para construir um sistema para a plataforma BeagleBone Black. No entanto, como queremos aprender sobre o Buildroot, iniciaremos nossa própria configuração do zero. Inicie a configuração do Buildroot:

```
$ make menconfig
```

Claro, você pode experimentar as outras ferramentas de configuração nconfig, xconfig ou config. Agora vamos fazer a configuração:

- menu **Target options**
  - Como a BeagleBone Black é uma plataforma baseada em ARM, então selecione **ARM (little endian)** em **Target Architecture**.
  - Beaglebone usa um AM335x da TI, baseado no núcleo ARM Cortex-A8. Portanto, selecione o **cortex-A8** em **Target Architecture Variant ()**.
  - No ARM, duas Interfaces Binárias de Aplicativos estão disponíveis: **EABI** e **EABIhf**. A menos que você tenha problemas de compatibilidade com binários pré-criados, o em **Target ABI** (que já deve ser o padrão de qualquer maneira).
  - Configurar **VFPv3-D16** em **Floating point strategy** é um padrão sensato de usar.

  - Os outros parâmetros podem ser deixados com seu valor padrão: em **ARM instruction set** o padrão é usar **ARM** (podemos usar o Conjunto de instruções **Thumb-2** para código um pouco mais compacto) e **ELF** é o único formato binário de destino disponível em **Target Binary Format**.

- Menu **Build options**

  - Não temos nada de especial para mudar neste menu, mas aproveite esta oportunidade para visitar este menu e ver as opções disponíveis. Cada opção tem um texto de ajuda que informa mais sobre a opção.

- Menu **Toolchain**

  - Por padrão, o Buildroot cria sua próprio toolchain. Isso leva um pouco de tempo e, para plataformas ARMv7, há um toolchain pré-criado fornecido pelo ARM. Selecione **External toolchain** em **Toolchain type** (). No entanto, não hesite em examinar as opções.
  - Selecione **Arm ARM 2021.07** em **Toolchain** (). O Buildroot pode usar toolchain predefinidas, como as fornecidas pela ARM, ou toolchain personalizadas (baixadas de um determinado local ou pré-instaladas em sua máquina).

- Menu **System configuration**

  - Para nosso sistema básico, não precisamos de muita configuração de sistema personalizada no momento. Portanto, reserve um tempo para examinar as opções disponíveis e coloque alguns valores personalizados para o hostname do sistema, o banner do sistema e a senha de root.

- Menu **Kernel**

  - Obviamente, precisamos de um kernel para rodar em nossa plataforma, então habilite a opção **Linux kernel**.
  - Por padrão, é usada a versão mais recente do kernel disponível no momento do lançamento do Buildroot. No nosso caso, queremos usar a mesma versão do git usado anteriormente. Portanto, selecione **Custom Git repository** no **Kernel version** e insira o link em **URL of custom repository**.
  - Agora, precisamos definir qual configuração de kernel usar. Começaremos usando uma configuração padrão fornecida nos fontes do kernel, chamada defconfig. Na prática, para esta plataforma defina o arquivo de configuração em **Defconfig name**.
  - O **Kernel binary format** é a próxima opção. Como vamos usar um gerenciador de inicialização U-Boot recente, manteremos o padrão do formato **uImage**.
  - No ARM, todas as plataformas modernas agora usam Device Tree para descrever o hardware. O BeagleBone Black está nessa situação, então você terá que habilitar a opção **Build a Device Tree Blob** e definir o nome em **In-tree Device Tree Source file names**. Mesmo que falar sobre Device Tree esteja além do escopo desta prática, sinta-se à vontade para dar uma olhada neste arquivo para ver o que ele contém.
  - A configuração do kernel para esta plataforma requer que o OpenSSL esteja disponível na máquina host. Para evitar depender dos arquivos de desenvolvimento OpenSSL instalados pela distribuição Linux de sua máquina host, o Buildroot pode construir sua própria versão: basta habilitar a opção **Needs host OpenSSL**.

- Menu **Target packages**
  - Este é provavelmente o menu mais importante, pois é onde você pode selecionar 2800+ pacotes Buildroot disponíveis, quais devem ser compilados e instalados em seu sistema. Para nosso sistema básico, habilitar o **BusyBox** é suficiente e já vem habilitado por padrão, mas fique à vontade para explorar os pacotes disponíveis.
- Menu **Filesystem images**
  - Por enquanto, mantenha apenas a opção **tar the root filesystem** habilitada. Cuidaremos separadamente de gravar o sistema de arquivos no SDcard.
- Menu **Bootloaders**.
  - Usaremos o bootloader ARM mais popular, habilitando **U-Boot**.
  - Selecione **Kconfig** em **Build system**. O U-Boot está em transição de uma situação em que todas as plataformas de hardware foram descritas em arquivos de cabeçalho C para um sistema em que o U-Boot reutiliza a lógica de configuração do kernel do Linux.
  - Use **Custom Git repositor** em **U-Boot Version ()**.
  - Na opção **URL of custom repository** insira o link do git usado anteriormente.
  - Em **Custom repository version** insira a versão que vocês usou.
  - Para muitas plataformas AM335x, o U-Boot tem uma única configuração, que pode então receber a plataforma de hardware exata para suportar o uso de um Device Tree. Portanto, precisamos habilitar **Using an in-tree board defconfig file** em **U-Boot configuration**, e definir o arquivo defconfig para a Beaglebone Black em **Board defconfig**.
  - O U-Boot no AM335x é dividido em duas partes: a inicialização de primeiro estágio e segundo estágio. Portanto, selecione **u-boot.img** em **U-Boot binary format**, e também habilite **Install U-Boot SPL binary image**, em seguida escreva **MLO** em **U-Boot SPL/TPL binary image name(s)**.

Agora terminamos a configuração básica do sistema para beaglebone black!

3 Build

Você poderia simplesmente executar make, mas como gostaríamos de manter um log da compilação, redirecionaremos as saídas padrão e de erro para um arquivo, bem como o terminal usando o comando **tee**:

```
$ make 2>&1 | tee build.log
```

Enquanto a compilação estiver em andamento, consulte as seções a seguir para preparar o que será necessário para testar os resultados da compilação.

4 Gravando o Sistema

Depois que o Buildroot terminar de construir o sistema, é hora de colocá-lo no cartão SD:

1. Copie os arquivos MLO, u-boot.img, zImage e am335x-boneblack.dtb de output/images/ para a partição de inicialização do SDcard.
2. Extraia o arquivo rootfs.tar para a partição rootfs do servirdor NFS, usando:

```
$ sudo tar -C /XXX/rootfs/ -xf output/images/rootfs.tar
```

Desmonte a partição do SDcard de forma limpa e ejete o SDcard.

4.1 Testando a conexão SSH

Execute os comandos abaixo no target para que o servidor SSH permita o login do usuário root sem senha:

```
# echo 'PermitRootLogin yes' >> /etc/ssh/sshd_config
# echo 'PermitEmptyPasswords yes' >> /etc/ssh/sshd_config
# /etc/init.d/S50sshd restart
```

Agora abra um terminal no host e teste a conexão SSH com o target:

```
$ ssh root@10.1.1.2
```

Você deverá ter acesso remoto ao target via SSH.

5 Atividades Práticas

pratica 1

Personalize o seu sistema no buildroot (ex. login, hostname, etc) para realizar uma inicialização proprietária e caso não tenha servidor ssh rodando inclua no sistema.

pratica 2

A partir do servidor HTTP e da página WEB criados, acesse um script em shell ou python para Ligar ou desligar os 4 LEDs da placa via uma página WEB (você pode usar CGI ou PHP, ou node).

pratica 3

Gerar um sistema completo com kernel, device tree, sistema de arquivo e Aplicação que façam os 4 LEDs piscarem, e coloque o seu App e todos os serviços (ssh, htttp) para iniciarem no init.d do sistema Linux.