Buildroot
BASC2020 seminar

Giacomo Longo

University of Genoa

11 December 2020

Table of contents

BuildRoot

BuildRoot

What's BuildRoot

Official website: https://buildroot.org

- Born in 2005

Official website: https://buildroot.org

▶ Born in 2005

▶ Entirely based on makefiles and kconfig

BuildRoot

Official website: https://buildroot.org

- ▶ Born in 2005
- ▶ Entirely based on makefiles and kconfig
- ▶ Only one goal: *producing root file system images* for *100% custom Linux systems*

BuildRoot users

The most prominent users of BuildRoot are using it for building:

- ▶ IoT devices

BuildRoot users

The most prominent users of BuildRoot are using it for building:

- ▶ IoT devices
- ▶ Automated factory controllers

BuildRoot users

The most prominent users of BuildRoot are using it for building:

- ▶ IoT devices
- ▶ Automated factory controllers
- ▶ Point of sale devices

BuildRoot users

The most prominent users of BuildRoot are using it for building:

- ▶ IoT devices
- ▶ Automated factory controllers
- ▶ Point of sale devices
- ▶ Car multimedia units

BuildRoot users

The most prominent users of BuildRoot are using it for building:

- ▶ IoT devices
- ▶ Automated factory controllers
- ▶ Point of sale devices
- ▶ Car multimedia units
- ▶ High end Hi-Fi amplifiers

BuildRoot

Why BuildRoot

- Each buildroot is a 100% custom Linux "mini-distro"

Why BuildRoot

- Each buildroot is a 100% custom Linux "mini-distro"
- Buildroot images can be less than 100MB or even 10MB

Why BuildRoot

- Each buildroot is a 100% custom Linux "mini-distro"
- Buildroot images can be less than 100MB or even 10MB
- Complete customization of target architecture and build flags

Why BuildRoot

- Each buildroot is a 100% custom Linux "mini-distro"
- Buildroot images can be less than 100MB or even 10MB
- Complete customization of target architecture and build flags
- Multiple compiler / libc / system layout choices

Why BuildRoot

- ▶ Each buildroot is a 100% custom Linux "mini-distro"
- ▶ Buildroot images can be less than 100MB or even 10MB
- ▶ Complete customization of target architecture and build flags
- ▶ Multiple compiler / libc / system layout choices
- ▶ Updated every 3 months current version is 2020.11

Why BuildRoot

- Each buildroot is a 100% custom Linux "mini-distro"
- Buildroot images can be less than 100MB or even 10MB
- Complete customization of target architecture and build flags
- Multiple compiler / libc / system layout choices
- Updated every 3 months current version is 2020.11
- Easily extendable

Why BuildRoot: architecture support

≈ 20 architectures supported

- ARC LE & BE
- **ARM** LE & BE
- AArch64 LE & BE
- csky
- **i386**
- Microblaze AXI & Non-AXI
- MIPS LE & BE
- MIPS64 LE & BE
- nds32

- Nios II
- PowerPC
- PowerPC64 LE & BE
- RISCV
- SuperH
- SPARC
- **x86_64**
- Xtensa

BuildRoot

BuildRoot process

What the user sees

1. Create a configuration file

The BuildRoot process

What the user sees

1. Create a configuration file
2. Start the build

The BuildRoot process

What the user sees

1. Create a configuration file
2. Start the build

What BuildRoot does

1. Build a cross compiler on our machine
2. Resolve the configuration dependencies
3. Compile from source the requested packages
4. Assemble an image

The BuildRoot process

What the user sees

1. Create a configuration file
2. Start the build
3. Flash the image on the device

What BuildRoot does

1. Build a cross compiler on our machine
2. Resolve the configuration dependencies
3. Compile from source the requested packages
4. Assemble an image

Creating some BuildRoots

Creating some BuildRoots
Prerequisites

Prerequisites
Packages for an ARM BuildRoot

Ubuntu 20.04

```
sudo apt-get update
sudo apt-get install -y \
  curl tar \
  make \
  gcc g++ \
  libncurses-dev libssl-dev \
  qemu-user-static \
  qemu-system-arm
```

Others

Binaries needed

Downloaders curl & wget

Extractor tar

Compilers gcc & g++

Libraries ncurses & openssl

Execution QEMU system for
ARM & QEMU static

1. Clone the repository at
   https://github.com/gabibbo97/basc-buildroot
2. Enter the directory
3. Run sh ./seminar-scripts/get-buildroot.sh

Please use the provided script

The script downloads BuildRoot 2020.11 but also applies two required patches that we need for today's seminar
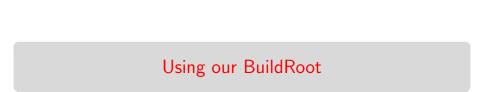
Creating some BuildRoots

Creating an ARM cross compiler

Creating some BuildRoots
Creating an ARM root filesystem

Creating some BuildRoots

Creating a bootable ARM root filesystem

Using our BuildRoot

Using our BuildRoot
Producing binaries for the target

Using the cross-compiler

1. Extract the cross-compiler
2. Run `relocate-sdk.sh`
3. Edit your `$PATH` variable: `export PATH="$PATH:$PWD/bin"`
4. You can invoke your cross compiler with commands like
   `arm-buildroot-linux-gnueabihf-<COMMAND NAME>`
   Notable entries
   - ▶ `arm-buildroot-linux-gnueabihf-gcc`
   - ▶ `arm-buildroot-linux-gnueabihf-gdb`
   - ▶ `arm-buildroot-linux-gnueabihf-nm`

Improving gdb with library simbols

See the section

Using our BuildRoot
Running dynamic executables

Running dynamic executables in Docker

```
sudo docker import rootfs.tar basc-buildroot
sudo docker run --rm -it \
  --volume "$(which qemu-arm-static):/bin/qemu-arm-static" \
  --volume "${PWD}/:/host" \
  --entrypoint /bin/qemu-arm-static \
  --workdir "/host" \
  basc-buildroot \
  /bin/sh
```

Running dynamic executables with systemd-nspawn

```
mkdir -p basc-rootfs
tar -xf rootfs.tar -C basc-rootfs
cp -f "$(which qemu-arm-static)" \
  basc-rootfs/bin/qemu-arm-static
sudo systemd-nspawn \
  --register=no \
  -D basc-rootfs \
  /bin/qemu-arm-static /bin/sh
```

Package needed

You might need to install the package `systemd-container`

Using our BuildRoot
Performing dynamic analysis

Booting the rootfs

```sh
#!/bin/sh
#
# Boots the built rootfs
#
exec qemu-system-arm \
  -machine virt \
  -cpu cortex-a7 \
  -smp 2 -m 2000 \
  -kernel bootable-rootfs/zImage \
  -device virtio-blk-device,drive=rootfs \
  -drive file=bootable-rootfs/rootfs.ext2,if=none,format=raw,id=rootfs \
  -append "console=ttyAMA0,115200 rootwait root=/dev/vda" \
  -netdev user,id=user0,hostfwd=tcp::2222-:22,hostfwd=tcp::1234-:1234 \
  -device virtio-net-device,netdev=user0 \
  -serial stdio \
  -display none
```

Opening an SSH session

```
exec ssh \
  -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no \
  -p 2222 \
  root@localhost
```

Sharing a folder

```
mkdir -p guest-os-ssh
exec sshfs root@localhost:/ ./guest-os-ssh \
  -f \
  -o port=2222 \
  -o reconnect \
  -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no
```

- {l,s,uf}trace do work as expected
- Can only be performed on QEMU **system** emulation

ltrace is buggy on ARM

ltrace has a bug with unwinding DWARF tables on ARM and will show limited information.
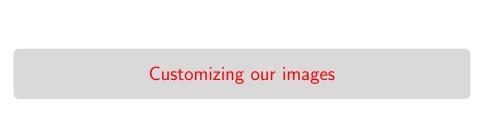
Using gdb

On the guest
gdbserver :1234 *command to debug*
On the host (*From the cross-compiler extracted folder*)[1]
 bin/_{arm-buildroot-linux-gnueabihf-}gdb \
  -x _{arm-buildroot-linux-gnueabihf}/sysroot/usr/share/buildroot/gdbinit \
  executable name
On the host gdb shell attach with target remote localhost:1234

---

[1]or use run-cross-gdb.sh from my release package

Customizing our images

- ▶ Create a directory
- ▶ Add `BR2_ROOTFS_OVERLAY=my-overlay` to `.config`
- ▶ Rebuild using `make`
- ▶ The structure of `my-overlay` will be copied to the rootfs

How to specify multiple overlays

Multiple overlays can be specified by separating them with spaces in the `BR2_ROOTFS_OVERLAY` directive

Add `BR2_ROOTFS_POST_BUILD_SCRIPT=my-script.sh` to `.config`
Available environment variables inside:

`BR2_CONFIG`	path of `.config`
`HOST_DIR`	path of `output/host`
`STAGING_DIR`	path of `output/staging`
`TARGET_DIR`	path of `output/target`
`BUILD_DIR`	path of `output/build`
`BINARIES_DIR`	path of `output/images`
`BASE_DIR`	path of `output`

How to specify multiple scripts
Multiple scripts can be specified by separating them with spaces in the
`BR2_ROOTFS_POST_BUILD_SCRIPT` directive

1. Add your files to the `output/target` directory
2. Rebuild using `make`

Warning

Your files might be rewritten / deleted by buildroot

Customizing our images

1. Unpack your rootfs (with `tar -xzf` for instance)
2. Perform your modifications
3. Repack your rootfs (with `tar -cf` for instance)