Building root filesystem, Linux Kernel and U-Boot for the Zynq with Buildroot

Pavol Kurina
pavol.kurina@gmail.com

FPGA-Kongress
München: 21.05.-23.05.2019

About me

- Independent software engineer
  - Freelancing 4 years now
- 15 years of professional experience
  - embedded systems, industrial vision, Linux, agile methods (TDD, CI/CD)
- Living in Ilmenau, Germany
- e-mail: pavol.kurina@gmail.com

Overview

- What is Buildroot?
- How does it compare to PetaLinux/Yocto?
- Getting started
- Customization options
- Managing customizations
- Summary

What is buildroot?

- From buildroot.org:

  Buildroot is a simple, efficient and easy-to-use tool to generate embedded Linux systems through cross-compilation.

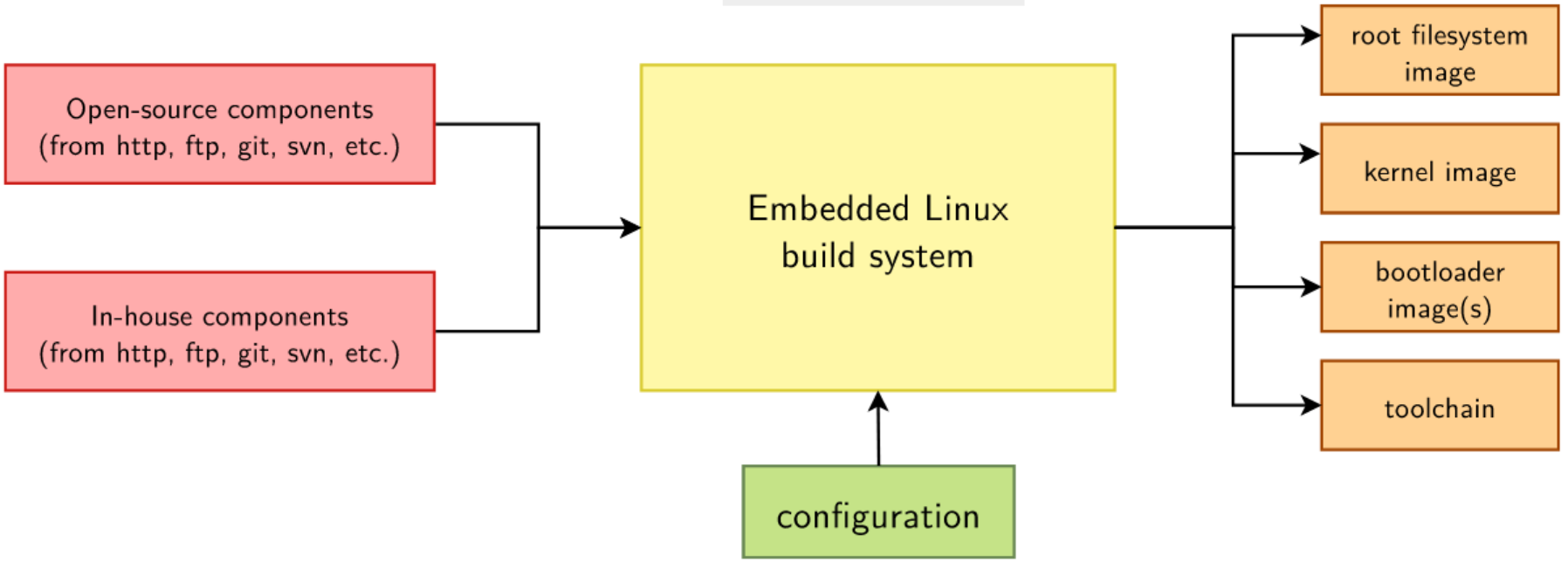
- From “The Buildroot user manual”:

  ... Buildroot is basically a set of Makefiles that download, configure, and compile software with the correct options.

- Popular boards supported out-of-the-box
- Provides packages for 2500+ applications/libraries
- Open-source with active community
- Regular releases and extensive documentation

How buildroot works?

1. Download
2. Configure
3. Build
4. Install



Adapted from https://bootlin.com/doc/training/buildroot/buildroot-slides.pdf, slide “Embedded Linux build system”

How does it compare to PetaLinux/Yocto?

- BR is designed for simplicity

  → faster to get started, smaller rootfs
- Yocto is designed for flexibility (layered system composition, partial updates)

  → steeper learning curve
- BR doesn’t support partial updates

  → firmware-builder vs. distribution-builder
- PetaLinux integrates better with Xilinx-SDK

Buildroot workflow

- Getting started from minimal configuration
- Customizing the configuration
- Integrating project-specific components
- Managing customizations for reproducible builds

Getting started (1)

- Download/clone Buildroot (~1min, ~150MB)

```
$ git clone https://git.buildroot.net/git/buildroot.git
$ cd buildroot
($ git checkout -b demo 2018.08)
```

- Which Zynq boards are supported?

```
buildroot$ make list-defconfigs | grep zynq

  zynq_microzed_defconfig              - Build for zynq_microzed
  zynqmp_zcu106_defconfig              - Build for zynqmp_zcu106
  zynq_zc706_defconfig                 - Build for zynq_zc706
  zynq_zed_defconfig                   - Build for zynq_zed
  zynq_zybo_defconfig                  - Build for zynq_zybo
```

Getting started (2)

- Configure Buildroot for our MicroZed board (<10sec)

```
buildroot$ make zynq_microzed_defconfig
```

- Build (~20min, 5.5GB)

```
buildroot$ make
```

- Update the SD-card (~10sec)

```
buildroot$ dd if=output/images/sdcard.img of=/dev/mmcblk0
```

- What did we get?

```
$ mount /dev/mmcblk0p1 /mnt/
/mnt/
├── boot.bin
├── devicetree.dtb
├── uImage
└── uramdisk
```

Let’s try to boot...

```
U-Boot SPL 2018.02 (April 02 2019 - 03:23:47)
spl_load_image_fat: error reading image fpga.bin, err - -2
spl_load_image_fat_os: error reading image system.dtb, err - -2
reading u-boot.img
reading u-boot.img

U-Boot 2018.02 (April 02 2019 - 03:23:47 +0200)

Model: Zynq MicroZED Board
...

Starting kernel ...

Linux version 4.9.0-xilinx (pavol@Latitude2) (gcc version 7.3.0
(Buildroot 2018.08) ) #1 SMP PREEMPT Thu April 2 02:03:21 CEST 2019
...

Welcome to Buildroot
buildroot login:
```

Buildroot workflow

- Getting started from minimal configuration
- **Customizing the configuration**
- Integrating project-specific components
- Managing customizations for reproducible builds

Customizing the configuration… (1)

- Example: enabling remote access
  - Enable and configure the network interface
  - Enable SSH server (dropbear)
- Using a **rootfs overlay**

```
buildroot/board/demo/
└── rootfs_overlay
    └── etc
        ├── dropbear
        │   └── ..._host_key
        └── network
            └── interfaces
```

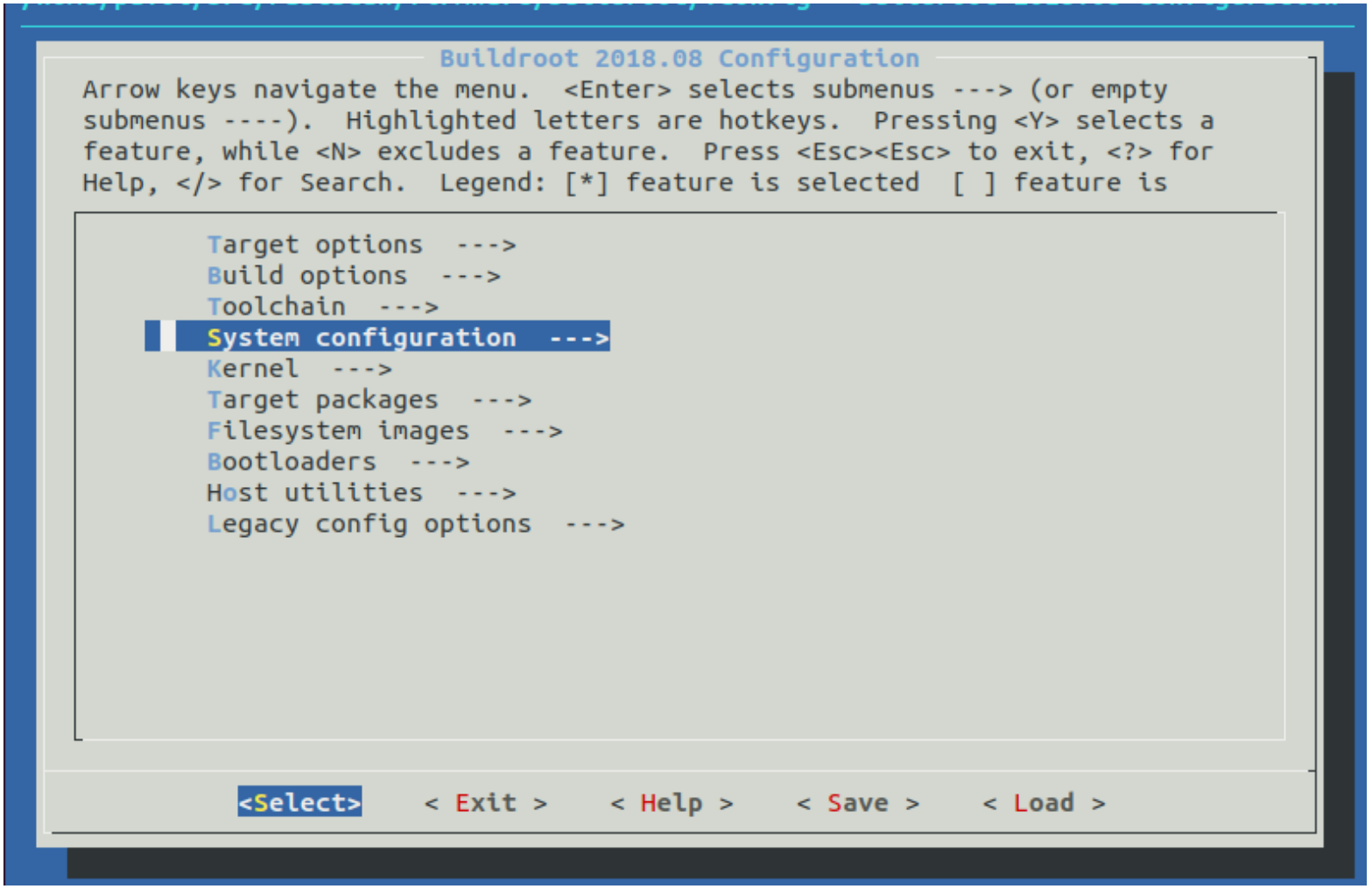
```
# /etc/network/interfaces

# Configure Loopback
auto lo
iface lo inet loopback

auto eth0
    iface eth0 inet dhcp
    hostname buildroot
```

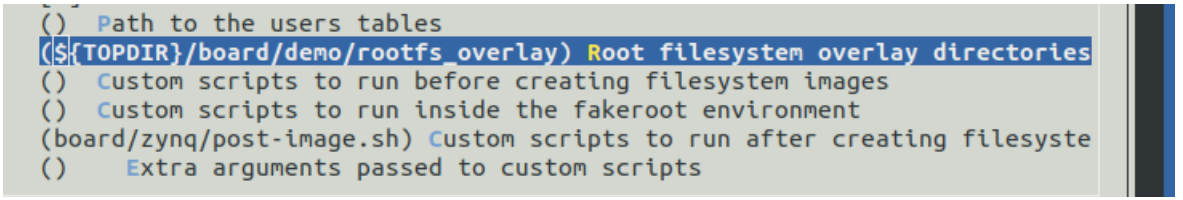
Customizing the configuration... (2)

```
buildroot$ make menuconfig
```

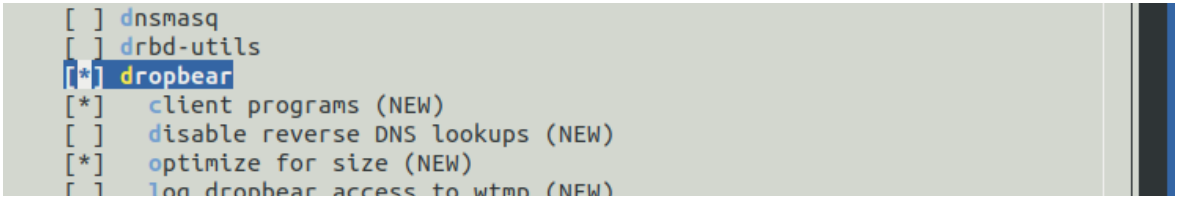


Customizing the configuration... (3)

- Tell Buildroot to use the rootfs overlay
  - System configuration → Root filesystem overlay…

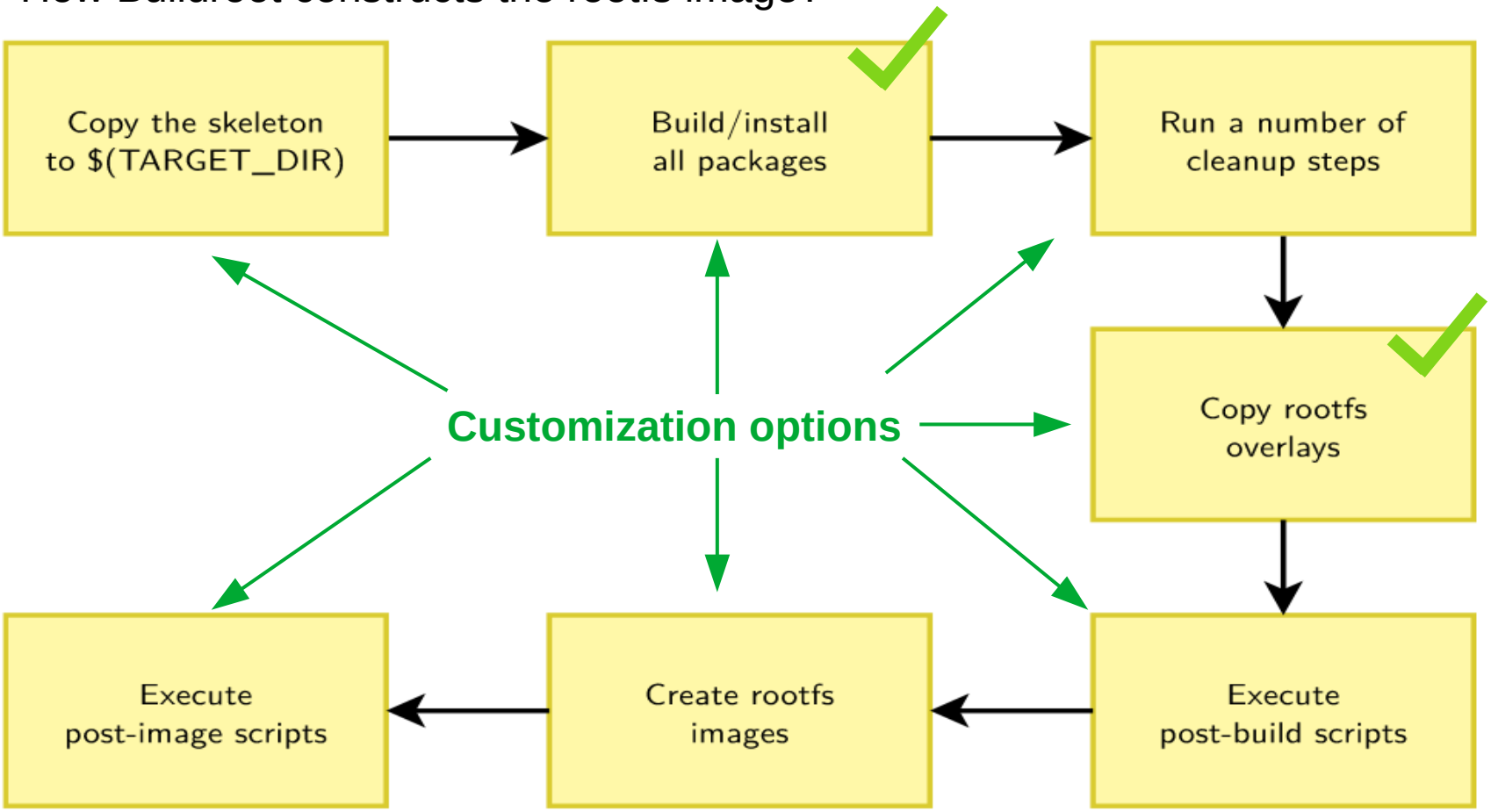


- Enable the “dropbear” package
  - Target packages → Networking applications → dropbear



Customization options

How Buildroot constructs the rootfs image?



Adapted from https://bootlin.com/doc/training/buildroot/buildroot-slides.pdf, slide “Overlall rootfs construction steps”

Buildroot workflow

- Getting started from minimal configuration
- Customizing the configuration
- **Integrating project-specific components**
  - Zynq: PL parts
  - Third-party libraries and applications
- Managing customizations for reproducible builds

Integrating the PL parts (1)

- Input from the “hardware team”:

  zynq_fsbl.elf, system_top.bit, u-boot.elf, system.dtb

- Add custom **post-image script**

```
buildroot/board/demo/
├── boot.bif
├── genimage.cfg
├── image.its
├── post-image.sh
├── pre-built
│   ├── system.dtb
│   ├── system_top.bit
│   ├── u-boot.elf
│   └── zynq_fsbl.elf
└── rootfs_overlay
```

```
...
echo "Creating boot.bin..."
mkbootimage boot.bif ${BINARIES_DIR}/boot.bin

echo "Creating FIT-image…"
mkimage -f image.its ${BINARIES_DIR}/image.ub
...

echo "Creating the sd-card image..."
support/scripts/genimage.sh -c genimage.cfg
```

Integrating the PL parts (2)

- Tell Buildroot to use the custom post-image script
  - System configuration → ...

```
(${TOPDIR}/board/demo/rootfs_overlay) Root filesystem overlay directories
(${TOPDIR}/board/demo/post-build.sh) Custom scripts to run before creating filesystem image
()  Custom scripts to run inside the fakeroot environment
(${TOPDIR}/board/demo/post-image.sh) Custom scripts to run after creating filesystem images
()    Extra arguments passed to custom scripts
```

- Build and update the SD-card

```
buildroot$ make && dd if=output/images/sdcard.img of=/dev/mmcblk0
```

- What did we get now?

```
$ mount /dev/mmcblk0p1 /mnt/
/mnt/
├── boot.bin
└── image.ub
```

Integrating third-party components … (1)

- Adding a new package (recipe)
  - Describing how to **download, configure, build and install** a component
  - Describing configuration and build dependencies
- Many download methods supported
  - git/github, http(s), ftp, rsync, local etc.
- Many build environments supported
  - autotools, CMake, python-setuptools/distutils etc.

Integrating third-party components … (2)

- Topic for a separate talk
- Extensive documentation
  - The Buildroot user manual, chapter 17
  - Thousands of examples within Buildroot

Buildroot workflow

- Getting started from minimal configuration
- Customizing the configuration
- Integrating project-specific components
- **Managing customizations for reproducible builds**

Managing customizations (1)

- Saving the configuration

```
buildroot$ make savedefconfig \
                BR2_DEFCONFIG=demo_zynq_microzed_defconfig
```

- What we have now?

```
buildroot$ git status

Untracked files:
...
    board/demo/
    configs/demo_zynq_microzed_defconfig
```

- **Best practice:** Keeping customizations outside of Buildroot

Managing customizations (2)

Creating a “demo” project overlay for buildroot:

```
buildroot_demo/
├── .git
├── board                                   # all board customizations
│   └── demo
│       ├── ...
│       ├── post-image.sh
│       ├── pre-built
│       │   └── ...
│       └── rootfs_overlay
│           └── ...
├── configs                                 # configurations
│   └── demo_zynq_microzed_defconfig
│
├── packages                                # third-party package recipes
│   └── xilinx_axidma
│       ├── Config.in
│       └── xilinx_axidma.mk
├── Config.in                               # Buildroot-specific
├── external.desc                           # see user manual, chap. 9.2
└── external.mk                             #
```

Managing customizations (3)

Setting-up and using the “demo” project overlay:

```
$ git clone https://github.com/pavolk/buildroot_demo.git
$ cd buildroot_demo
$ mkdir build
$ cd build

$ make -C ${BUILDROOT_INSTALL_PREFIX} O=$(pwd) BR2_EXTERNAL=$(pwd)/.. \
         demo_zynq_microzed_defconfig

$ make menuconfig

$ make && dd if=images/sdcard.img of=/dev/mmcblk0

$ make savedefconfig
```

Summary

- Buildroot is usable for Zynq-boards
  - It is more lightweight than PetaLinux
- Mixed workflow Xilinx-Vivado/SDK + Buildroot is working and straight forward
  - Xilinx-Vivado/SDK → boot.bin
  - Buildroot → Kernel and rootfs → image.ub (FIT)
- Especially useful when switching from/to a board with Buildroot-based BSP (e.g. enclustra)

What I didn’t talk about?

- How to customize the kernel, u-boot and busybox configurations?
- When a clean build is necessary and why?
- How to re-build a single package?
- How to analyze BR with graphs?
- How to use BR as a cross-development environment with Eclipse/CDT?

References (1)

- Buildroot vs. OpenEmbedded/Yocto Project: A Four Hands Discussion
  - https://www.yoctoproject.org/learn-items/elc-2016-buildroot-vs-yocto-projectopenembedded-alexandre-belloni/
- The Buildroot user-manual
  - https://buildroot.org/docs.html
- bootlin: Buildroot training (slides)
  - http://bootlin.com/doc/training/buildroot/buildroot-slides.pdf
- Buildroot Eclipse-plugin Wiki
  - https://github.com/mbats/eclipse-buildroot-bundle/wiki

References (2)

- mkbootimage: open-source replacement for bootgen
  - https://github.com/antmicro/zynq-mkbootimage
- Xilinx: How to build a FIT-image
  - https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842374/U-Boot+Images
- “demo” project’s github-repository
  - https://github.com/pavolk/buildroot_demo.git

Thank you!

Any questions?

pavol.kurina@gmail.com

A post-build script

- Example:
  - Create */boot* directory within rootfs
  - Extend the */etc/fstab* to automatically mount the boot-partition

```
$ cat board/demo/post-build.sh

#!/bin/sh

BOARD_DIR="$(dirname $0)"
TARGET_DIR=$1

# Create /boot mountpoint, and adjust /etc/fstab
mkdir -p $TARGET_DIR/boot
echo "/dev/mmcblk0p1\t\t/boot\tvfat\tdefaults\t\t0\t0" >> $TARGET_DIR/etc/fstab
```

Adding new package

- Example: **xilinx_axidma**
  - `https://github.com/bperez77/xilinx_axidma.git`
  - driver, library and examples (benchmark)
- Adding package directory:

```
buildroot/package/xilinx_axidma/
├── Config.in                       # configuration options/dependencies
│
└── xilinx_axidma.mk                # steps to download, configure,
                                    # build and install the package;
                                    # build dependencies
```

Recipe

```
XILINX_AXIDMA_VERSION = 42ed91e
XILINX_AXIDMA_SITE = https://github.com/bperez77/xilinx_axidma.git
XILINX_AXIDMA_SITE_METHOD = git
XILINX_AXIDMA_INSTALL_STAGING = YES
XILINX_AXIDMA_INSTALL_TARGET = YES
XILINX_AXIDMA_MODULE_SUBDIRS = driver
XILINX_AXIDMA_MODULE_MAKE_OPTS = EXTRA_FLAGS="-I$(@D)/include"
...
define XILINX_AXIDMA_BUILD_CMDS
    $(TARGET_MAKE_ENV) $(MAKE) -C $(@D) library examples
endef

define XILINX_AXIDMA_INSTALL_STAGING_CMDS
    $(INSTALL) -D -m 644 $(@D)/include/*.h  $(STAGING_DIR)/usr/include
    $(INSTALL) -D -m 644 $(@D)/library/libaxidma.so $(STAGING_DIR)/usr/lib
endef

define XILINX_AXIDMA_INSTALL_TARGET_CMDS
    $(INSTALL) -D -m 644 $(@D)/library/libaxidma.so $(TARGET_DIR)/usr/lib
endef

$(eval $(kernel-module))
$(eval $(generic-package))
```

Using the recipe

```
$ make xilinx_axidma
```

```
>>> xilinx_axidma 42ed91e Downloading
>>> xilinx_axidma 42ed91e Extracting
>>> xilinx_axidma 42ed91e Building kernel module(s)
>>> xilinx_axidma 42ed91e Installing to staging directory
>>> xilinx_axidma 42ed91e Installing to target
>>> xilinx_axidma 42ed91e Installing kernel module(s)
```

```
$ find target/ -name "*axidma*"
```

```
target/usr/lib/libaxidma.so
target/lib/modules/4.6.0-xilinx-apf/extra/axidma.ko
```

```
$ find staging/ -name "*axidma*"
```

```
staging/usr/lib/libaxidma.so
staging/usr/include/libaxidma.h
staging/usr/include/axidma_ioctl.h
```