Bringing RAUC A/B Updates to More Linux Devices

Leon Anavi
Konsulko Group
leon.anavi@konsulko.com
leon@anavi.org
FOSDEM 2022

Konsulko Group

Agenda

- Software A/B updates with RAUC
- Integrating RAUC on embedded Linux Devices with Yocto and OpenEmbedded
- Examples
- Conclusions
- Q&A

Embedded Linux

Embedded Linux devices are all around the world… and even on Mars

Two things are needed to create a custom Linux distribution for embedded devices:

- Build system
- Software update mechanism

Are there any open source update solutions?

- Mender
- **RAUC**
- SWUpdate
- Swupd
- UpdateHub
- Balena
- Snap
- OSTree
- Aktualizr
- Aktualizr-lite
- QtOTA
- Torizon
- FullMetalUpdate
- Rpm-ostree (used in Project Atomic)

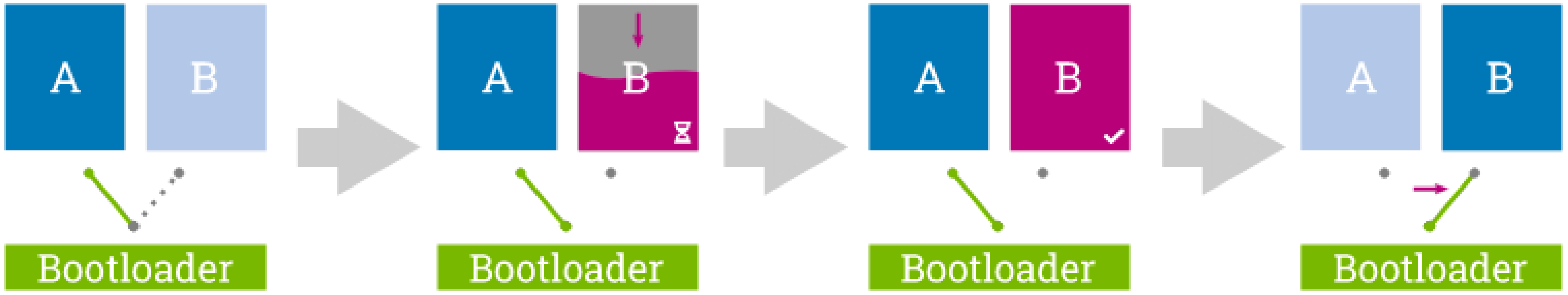
Common Embedded Linux Update Strategies

- A/B updates with dual redundant scheme
- Delta updates
- Container-based updates
- Combined strategies

What is RAUC?

- A lightweight update client that runs on an Embedded Linux device and reliably controls the procedure of A/B updating the device with a new firmware revision
- Provides tools for the build system to create, inspect and modify update bundles
- Uses X.509 cryptography to sign update bundles
- Compatible with the **Yocto Project and OpenEmbedded**, Buildroot and PTXdist
- Compatible with Eclipse hawkBit
- Started by Pengutronix in 2015,
  adopted by the community and the industry

How Does RAUC Work?



RAUC Licenses

- RAUC – LGPLv2.1
  https://github.com/rauc/rauc
- meta-rauc - MIT
  https://github.com/rauc/meta-rauc
- meta-rauc-community – MIT
  https://github.com/rauc/meta-rauc-community
- rauc-hawkbit – LGPLv2.1
  https://github.com/rauc/rauc-hawkbit
- rauc-hawkbit-updater – LGPLv2.1
  https://github.com/rauc/rauc-hawkbit-updater

Yocto/OpenEmbedded Layers for RAUC

- **meta-rauc**

  Layer for RAUC, the embedded Linux update framework

- **meta-rauc-community**

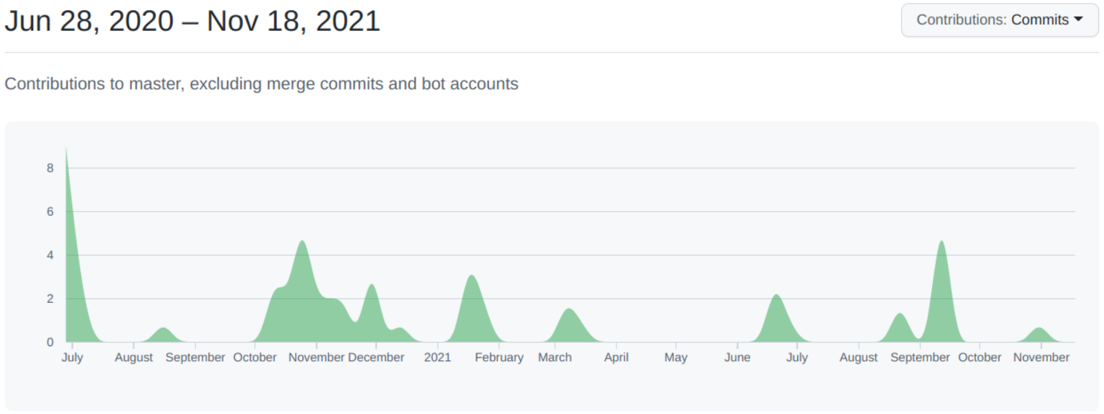
  Layer with examples for integration of RAUC, the embedded Linux A/B update framework

meta-rauc

- Yocto/OpenEmbedded meta layer for RAUC
- Supports all recent Yocto/OE releases: Honister, Gatesgarth, Dunfell, Zeus, Warrior, Thud, Sumo, Morty, Pyro and Krogoth
- Available under MIT license in GitHub: https://github.com/rauc/meta-rauc
- 33 contributors, the RAUC co-maintainer Enrico Jörns from Pengutronix is the leading contributor

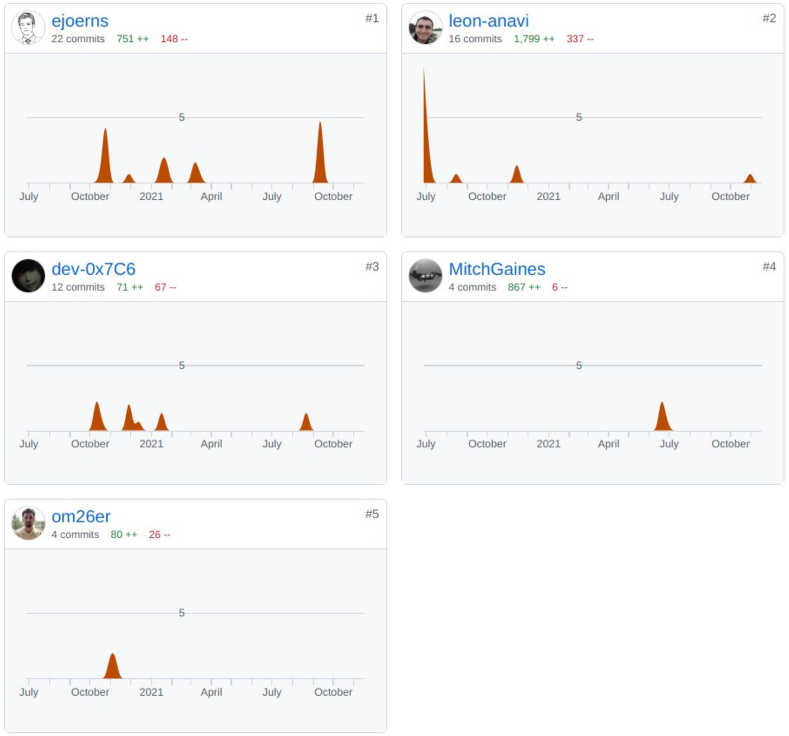
meta-rauc-community

- **meta-rauc-community** started in 2020
- Moved to RAUC GitHub organization in 2021
- 5 contributors



Jun 28, 2020 – Nov 18, 2021

Contributions to master, excluding merge commits and bot accounts



ejoerns #1
22 commits 751 ++ 148 --

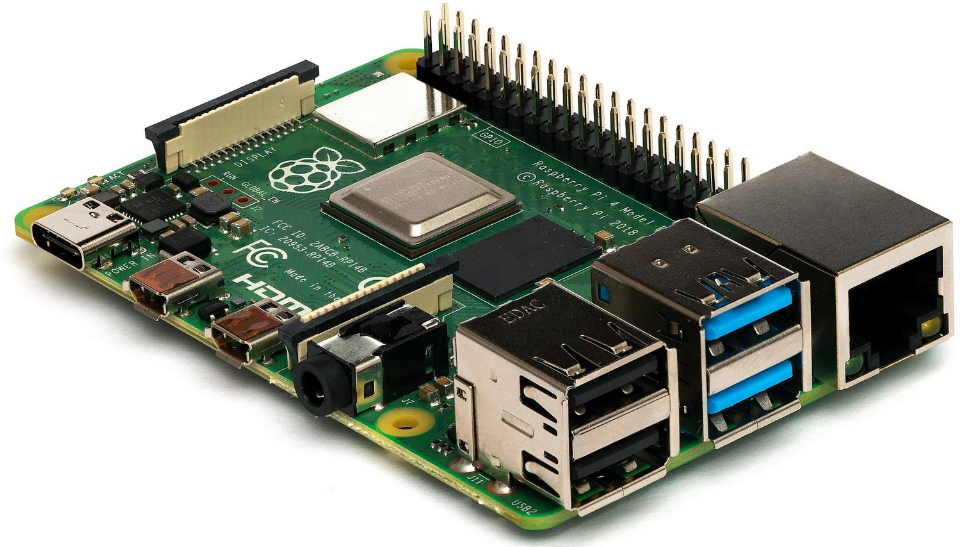
leon-anavi #2
16 commits 1,799 ++ 337 --

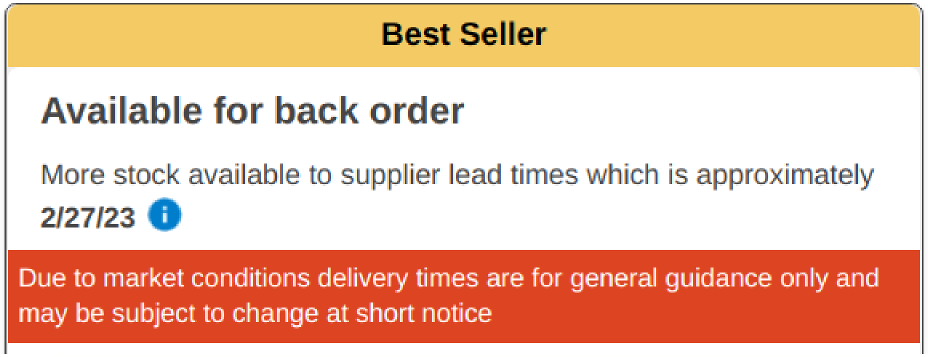
dev-0x7C6 #3
12 commits 71 ++ 67 --

MitchGaines #4
4 commits 867 ++ 6 --

om26er #5
4 commits 80 ++ 26 --

Raspberry Pi was the 1st machine in meta-rauc-community



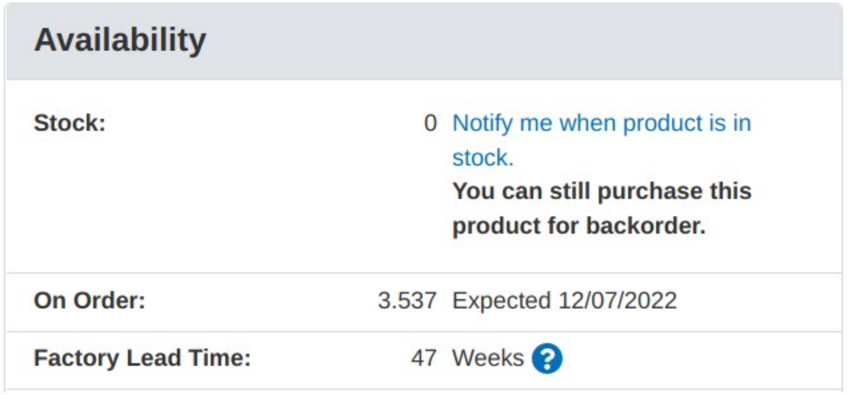


Best Seller

Available for back order

More stock available to supplier lead times which is approximately **2/27/23**

Due to market conditions delivery times are for general guidance only and may be subject to change at short notice



Availability

Stock:	0	Notify me when product is in stock. **You can still purchase this product for backorder.**
On Order:	3.537	Expected 12/07/2022
Factory Lead Time:	47	Weeks

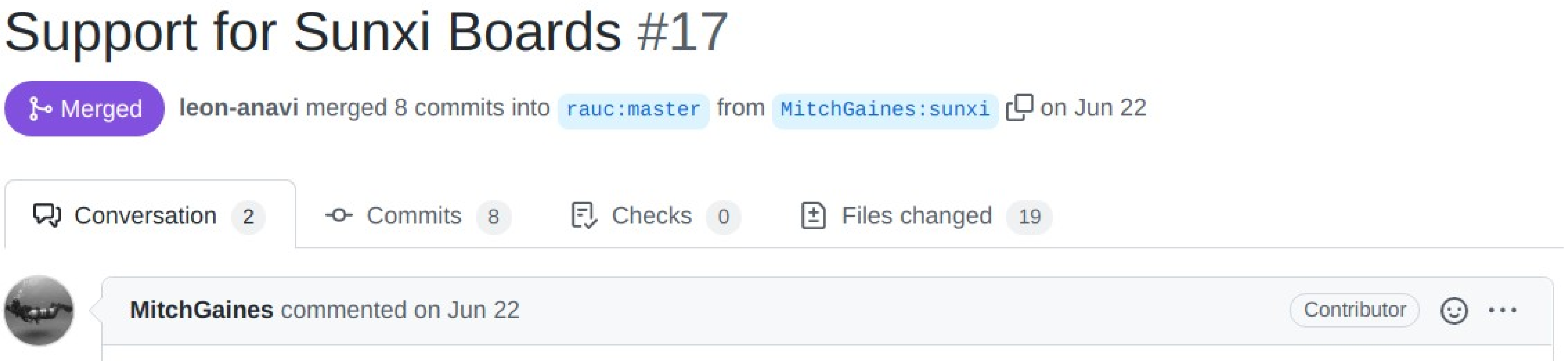
meta-rauc-community

Yocto/OpenEmbedded meta layer with demo examples for integration of RAUC, the embedded Linux A/B update framework:

- **meta-rauc-raspberrypi**: for Raspberry Pi
- **meta-rauc-qemux86**: for QEMU (qemux86-64)
- **meta-rauc-sunxi**: for Allwinner sunxi SoCs
- **meta-rauc-tegra**: for NVIDIA Jetson platforms, based on L4T

https://github.com/rauc/meta-rauc-community

Notable Contributions

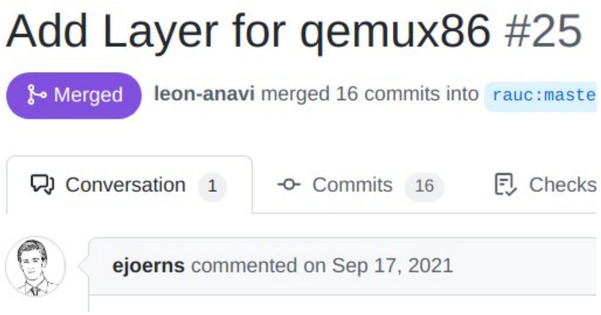


Support for Sunxi Boards #17

Merged leon-anavi merged 8 commits into rauc:master from MitchGaines:sunxi on Jun 22

Conversation 2 Commits 8 Checks 0 Files changed 19

MitchGaines commented on Jun 22 Contributor

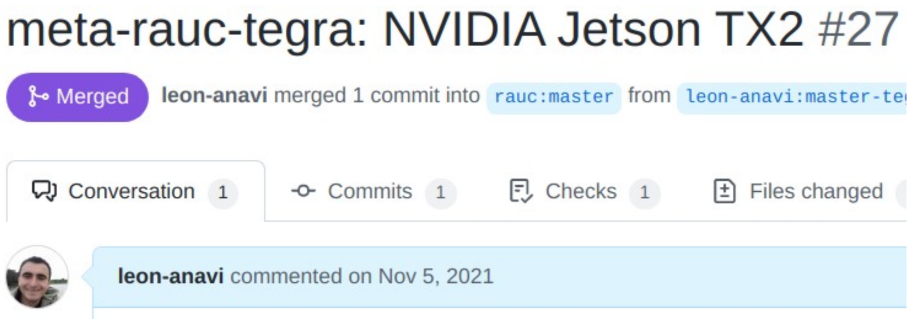


Add Layer for qemux86 #25

Merged leon-anavi merged 16 commits into rauc:maste

Conversation 1 Commits 16 Checks

ejoerns commented on Sep 17, 2021



meta-rauc-tegra: NVIDIA Jetson TX2 #27

Merged leon-anavi merged 1 commit into rauc:master from leon-anavi:master-te

Conversation 1 Commits 1 Checks 1 Files changed

leon-anavi commented on Nov 5, 2021

RAUC Integration Steps

- Select an appropriate bootloader
- Enable **SquashFS** in the Linux kernel configurations
- **ext4** root file system (RAUC does not have an ext2 / ext3 file type)
- Create specific partitions that match the RAUC slots
- Configure Bootloader environment and create a script to switch RAUC slots
- Create a certificate and a keyring to RAUC’s **system.conf**

RAUC Example with Raspberry Pi 4

- Download Poky, meta-openembedded and meta-raspberrypi:

```
git clone -b master git://git.yoctoproject.org/poky poky-rpi-rauc
cd poky-rpi-rauc
git clone -b honister git://git.openembedded.org/meta-openembedded
git clone -b honister git://git.yoctoproject.org/meta-raspberrypi
```

- Download RAUC related layers:

```
git clone -b honister https://github.com/rauc/meta-rauc.git
git clone -b honister https://github.com/rauc/meta-rauc-community.git
```

- Initialize the build environment:

```
source oe-init-build-env
```

RAUC Example with Raspberry Pi 4

- Add layers:

```
bitbake-layers add-layer ../meta-openembedded/meta-oe/
bitbake-layers add-layer ../meta-openembedded/meta-python/
bitbake-layers add-layer ../meta-openembedded/meta-networking/
bitbake-layers add-layer ../meta-openembedded/meta-multimedia/
bitbake-layers add-layer ../meta-raspberrypi/
bitbake-layers add-layer ../meta-rauc
bitbake-layers add-layer ../meta-rauc-community/meta-rauc-raspberrypi/
```

RAUC Example with Raspberry Pi 4

- Add to local.conf:

```
MACHINE = "raspberrypi4"
DISTRO_FEATURES:append = " systemd"
VIRTUAL-RUNTIME_init_manager = "systemd"
DISTRO_FEATURES_BACKFILL_CONSIDERED = "sysvinit"
VIRTUAL-RUNTIME_initscripts = ""
IMAGE_INSTALL:append = " rauc"
IMAGE_FSTYPES="tar.bz2 ext4 wic.bz2 wic.bmap"
SDIMG_ROOTFS_TYPE="ext4"
ENABLE_UART = "1"
RPI_USE_U_BOOT = "1"
PREFERRED_PROVIDER_virtual/bootloader = "u-boot"
WKS_FILE = "sdimage-dual-raspberrypi.wks.in"
```

RAUC Example with Raspberry Pi 4

- Build a minimal bootable image:

```
bitbake core-image-minimal
```

- Flash the image to a microSD card and boot it on Raspberry Pi 4:

```
sudo umount /dev/sdX*
bzcat tmp/deploy/images/raspberrypi4/core-image-minimal-raspberrypi4.wic.bz2 | sudo dd of=/dev/sdX
sync
```

- Attach USB to UART debug cable to Raspberry Pi 4, plug Ethernet cable and the microSD card. Turn on Raspberry Pi 4. Verify that the system boots successfully.

RAUC Update Bundle

- Add to conf/local.conf:

```
IMAGE_INSTALL:append = " nano"
```

- Build a RAUC bundle:

```
bitbake update-bundle
```

Manual RAUC Update of Raspberry Pi 4

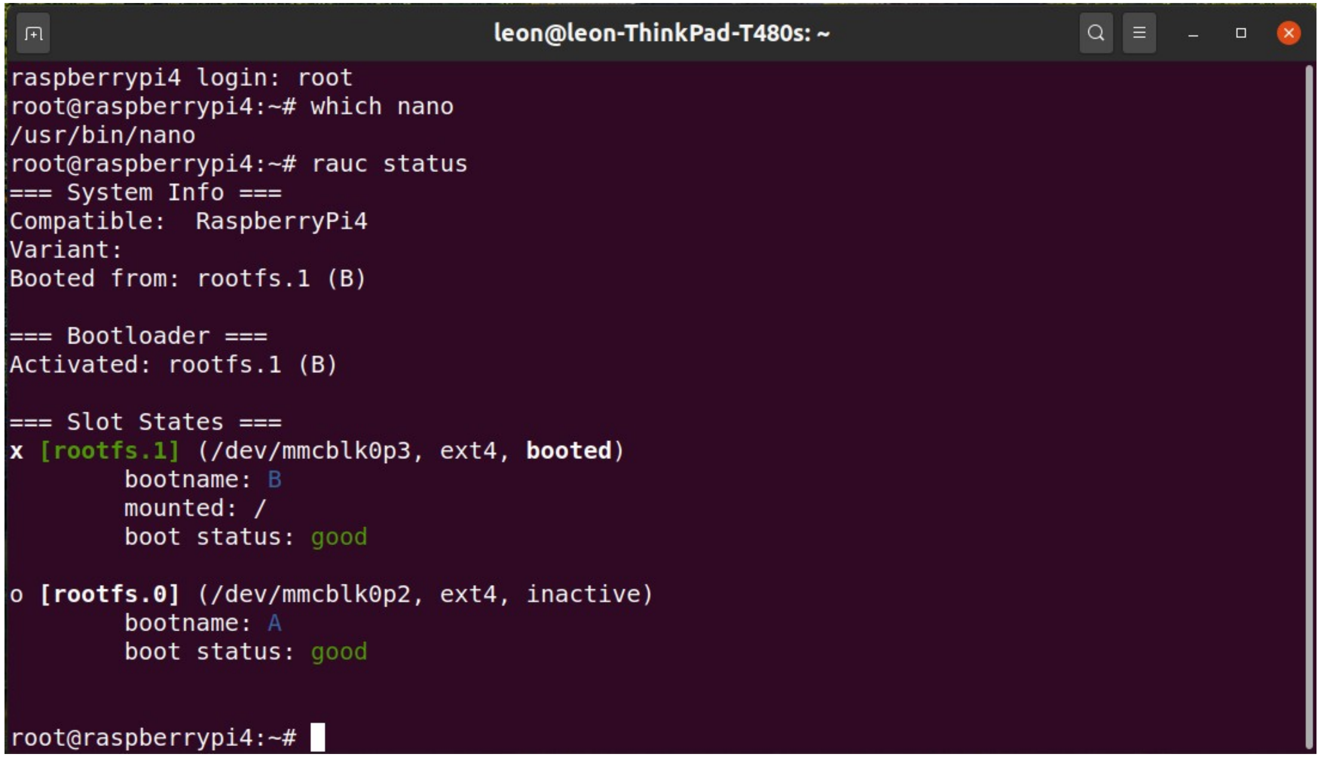
- On the build system:

```
cd tmp/deploy/images/raspberrypi4/
python3 -m http.server
```

- On the embedded device, in this case Raspberry Pi 4:

```
wget http://example.com:8000/update-bundle-raspberrypi4.raucb -P /tmp
rauc install /tmp/update-bundle-raspberrypi4.raucb
reboot
```

Check RAUC Status After Update



```
raspberrypi4 login: root
root@raspberrypi4:~# which nano
/usr/bin/nano
root@raspberrypi4:~# rauc status
=== System Info ===
Compatible:  RaspberryPi4
Variant:
Booted from: rootfs.1 (B)

=== Bootloader ===
Activated: rootfs.1 (B)

=== Slot States ===
x [rootfs.1] (/dev/mmcblk0p3, ext4, booted)
        bootname: B
        mounted: /
        boot status: good

o [rootfs.0] (/dev/mmcblk0p2, ext4, inactive)
        bootname: A
        boot status: good


root@raspberrypi4:~#
```

U-Boot Environment for RAUC

RAUC relies on the following U-Boot environment variables:

- BOOT_ORDER - a space-separated list of boot targets in the order they should be tried
- BOOT_<bootname>_LEFT - contains the number of remaining boot attempts to perform for the respective slot

- For details:

https://rauc.readthedocs.io/en/latest/integration.html#set-up-u-boot-environment-for-rauc

boot.cmd.in for RAUC & Raspberry Pi

```
fdt addr ${fdt_addr} && fdt get value bootargs /chosen bootargs
test -n "${BOOT_ORDER}" || setenv BOOT_ORDER "A B"
test -n "${BOOT_A_LEFT}" || setenv BOOT_A_LEFT 3
test -n "${BOOT_B_LEFT}" || setenv BOOT_B_LEFT 3
test -n "${BOOT_DEV}" || setenv BOOT_DEV "mmc 0:1"
setenv bootpart
setenv raucslot
for BOOT_SLOT in "${BOOT_ORDER}"; do
  if test "x${bootpart}" != "x"; then
    # skip remaining slots
  elif test "x${BOOT_SLOT}" = "xA"; then
    if test ${BOOT_A_LEFT} -gt 0; then
      setexpr BOOT_A_LEFT ${BOOT_A_LEFT} – 1
      echo "Found valid RAUC slot A"
      setenv bootpart "/dev/mmcblk0p2"
      setenv raucslot "A"
      setenv BOOT_DEV "mmc 0:2"
    fi
  elif test "x${BOOT_SLOT}" = "xB"; then
    if test ${BOOT_B_LEFT} -gt 0; then
      setexpr BOOT_B_LEFT ${BOOT_B_LEFT} – 1
      echo "Found valid RAUC slot B"
      setenv bootpart "/dev/mmcblk0p3"
      setenv raucslot "B"
      setenv BOOT_DEV "mmc 0:3"
    fi
  fi
done
```

```
if test -n "${bootpart}"; then
  setenv bootargs "${bootargs} root=${bootpart} rauc.slot=${raucslot}"
  saveenv
else
  echo "No valid RAUC slot found. Resetting tries to 3"
  setenv BOOT_A_LEFT 3
  setenv BOOT_B_LEFT 3
  saveenv
  reset
fi
fatload mmc 0:1 ${kernel_addr_r} @@KERNEL_IMAGETYPE@@
if test ! -e mmc 0:1 uboot.env; then saveenv; fi;
@@KERNEL_BOOTCMD@@ ${kernel_addr_r} - ${fdt_addr}
```

Generate RAUC Certificate

Use script openssl-ca.sh from meta-rauc to to create a certificate and a key:

- The target RAUC package must use the generated keyring file
- RAUC bundle recipe must use the generated key and certificate

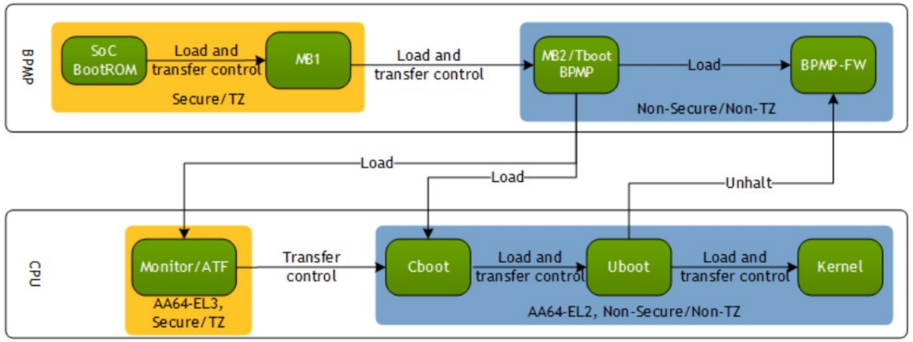
- For details: https://github.com/rauc/meta-rauc/blob/master/scripts/README

RAUC Bundle Generator update-bundle.bb

```
DESCRIPTION = "RAUC bundle generator"

inherit bundle

RAUC_BUNDLE_COMPATIBLE = "RaspberryPi4"
RAUC_BUNDLE_VERSION = "v20200703"
RAUC_BUNDLE_DESCRIPTION = "RAUC Demo Bundle"
RAUC_BUNDLE_SLOTS = "rootfs"
RAUC_SLOT_rootfs = "core-image-minimal"
RAUC_SLOT_rootfs[fstype] = "ext4"

RAUC_KEY_FILE = "${THISDIR}/files/development-1.key.pem"
RAUC_CERT_FILE = "${THISDIR}/files/development-1.cert.pem"
```

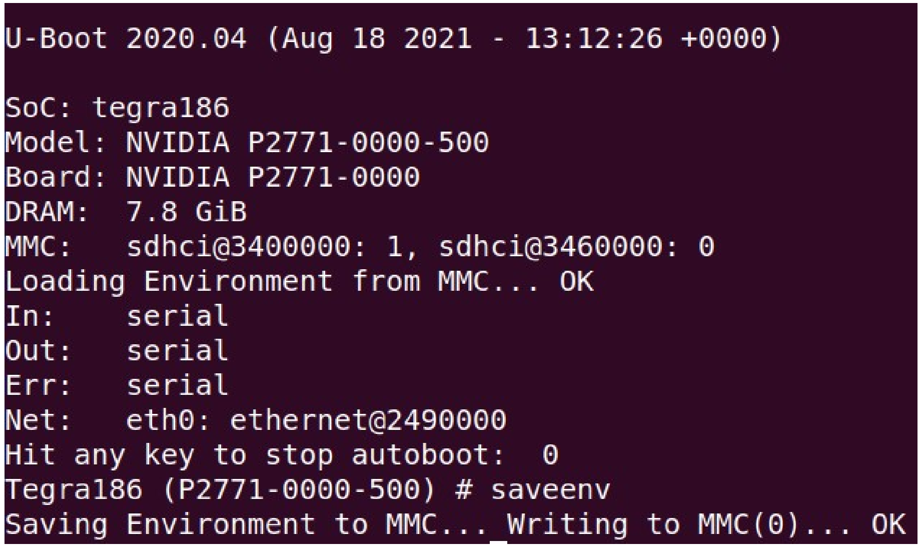
RAUC on NVIDIA Jetson TX2

- An example RAUC integration has been added for NVIDIA Jetson TX2: https://github.com/rauc/meta-rauc-community/tree/master/meta-rauc-tegra
- Based on Yocto/OE BSP **meta-tegra**: https://github.com/OE4T/meta-tegra
- Boot flow: Cboot > U-Boot > Kernel
- Patched include/configs/p2771-0000.h through u-boot-tegra to enable RAUC

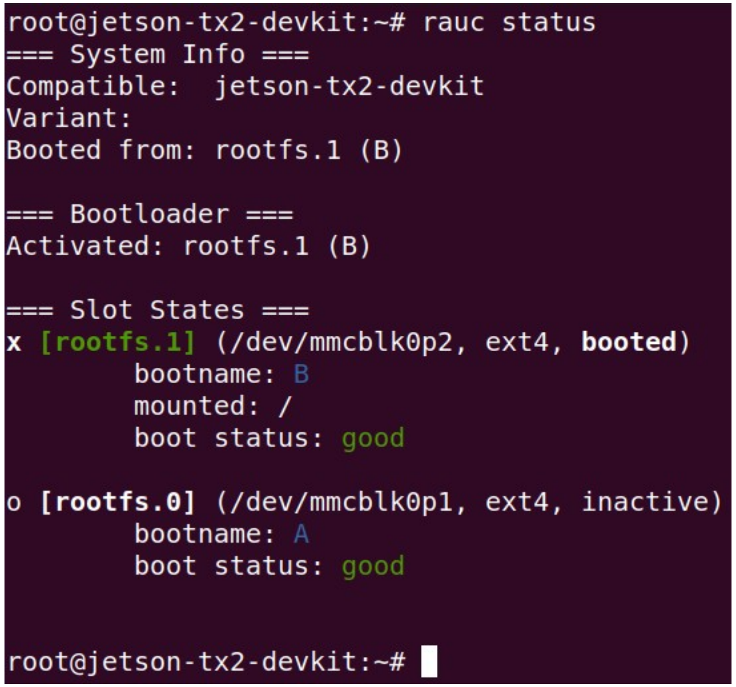


RAUC on NVIDIA Jetson TX2

- For the demo the U-Boot env is saved to MMC beware of conflicts for atomic bootloader updates



```
U-Boot 2020.04 (Aug 18 2021 - 13:12:26 +0000)

SoC: tegra186
Model: NVIDIA P2771-0000-500
Board: NVIDIA P2771-0000
DRAM:  7.8 GiB
MMC:   sdhci@3400000: 1, sdhci@3460000: 0
Loading Environment from MMC... OK
In:    serial
Out:   serial
Err:   serial
Net:   eth0: ethernet@2490000
Hit any key to stop autoboot:  0
Tegra186 (P2771-0000-500) # saveenv
Saving Environment to MMC... Writing to MMC(0)... OK
```



```
root@jetson-tx2-devkit:~# rauc status
=== System Info ===
Compatible:  jetson-tx2-devkit
Variant:
Booted from: rootfs.1 (B)

=== Bootloader ===
Activated: rootfs.1 (B)

=== Slot States ===
x [rootfs.1] (/dev/mmcblk0p2, ext4, booted)
        bootname: B
        mounted: /
        boot status: good

o [rootfs.0] (/dev/mmcblk0p1, ext4, inactive)
        bootname: A
        boot status: good


root@jetson-tx2-devkit:~#
```

SquashFS & Loopback Device Support

- To install RAUC bundles the kernel used on the embedded device must support both loop block devices and the SquashFS file system
- For example in linux-tegra_%.bbappend with a kernel configuration fragment:

```
FILESEXTRAPATHS:prepend := "${THISDIR}/files:"
SRC_URI += "file://rauc.cfg"
```

```
CONFIG_MD=y
CONFIG_BLK_DEV_DM=y
CONFIG_BLK_DEV_LOOP=y
CONFIG_DM_VERITY=y
CONFIG_SQUASHFS=y
CONFIG_CRYPTO_SHA256=y
CONFIG_SQUASHFS_FILE_DIRECT=y
SQUASHFS_DECOMP_MULTI=y
CONFIG_SQUASHFS_ZLIB=y
CONFIG_SQUASHFS_FRAGMENT_CACHE_SIZE=3
```

Conclusions

- RAUC is a secure, reliable, free and open source framework for A/B software updates of embedded Linux devices
- **meta-rauc-community** is the Yocto/OpenEmbedded layer providing RAUC example integration on popular embedded devices
- As of the moment **meta-rauc-community** provides examples for Raspberry Pi, QEMU x86-64, Allwinner (SunXi) and NVIDIA Jetson Tegra TX2
- Contributors wanted to extend the RAUC example integration on more embedded Linux devices

Thank You!

Useful links

- Software Updates with RAUC, the Yocto Project and OpenEmbedded, Leon Anavi Yocto Project Summit 2020
https://pretalx.com/yocto-project-summit-2020/talk/JJYPH3/
- Getting Started with RAUC on Raspberry Pi, an article at konsulko.com
https://www.konsulko.com/getting-started-with-rauc-on-raspberry-pi-2/
- Behind the Scenes of an Update Framework: RAUC, Enrico Jörns, ELCE 2019
https://www.youtube.com/watch?v=ZkumnNsWczM
- Embedded Recipes 2019 - Remote update adventures with RAUC, Yocto and Barebox
https://www.youtube.com/watch?v=hS3Fjf7fuHM
- Secure and Safe Updates for Your Embedded Device, Enrico Jörns, FOSDEM 2017
https://archive.fosdem.org/2017/schedule/event/secure_safe_embedded_updates/