From Zero to A/B

Swimming Upstream with

yocto project,  

ROL BOX & RAUC

Roland Hieber – r.hieber@pengutronix.de
Ahmad Fatoum – a.fatoum@pengutronix.de
Abstract

Many embedded projects start with the silicon vendor's BSP. After all, they have taken care of integrating everything, so what's not to love? But deviating from upstream comes with its own costs: Vendor BSPs are often full of non-upstreamable patches, outdated components, and hardships keeping the software up-to-date.

But mainline support for many SoCs is already at a level where it can be readily used. So how do you get to a point where a bootloader boots an upstream Linux installation, which can be updated in a redundant A/B fashion, all from within a maintainable and reproducible board support package?

In their talk, Roland and Ahmad will guide attendees through evaluating upstream SoC support in the low-level components with regards to project requirements, show how to integrate barebox as a bootloader into a Yocto BSP, and configure it to enable reliable, i.e. atomic and redundant, updates using RAUC.
About Us

- Roland Hieber
- Pengutronix e.K.
- rohieb
- r.hieber@pengutronix.de

- Kernel and boot loader porting
- Driver development
- System integration
- Embedded Linux Consulting
About Me #2

Ahmad Fatoum
Pengutronix e.K.
a3f
a.fatoum@pengutronix.de

- Kernel and boot loader porting
- Driver development
- System integration
- Embedded Linux Consulting
Downstream BSP Use

- Downstream vendor BSPs can be attractive
  - Work out-of-the-box on evaluation HW
  - Hopefully supported by vendor
  - Vendor-approved use cases covered
  - Available earlier

- But priorities will often diverge
  - Vendor will shift focus to newer hardware
  - Users will want to update

Why OS Updates?

- Embedded devices can have quite long lifespans. Operating system needs updates for
  - Patching security issues
  - Inter-operation with new devices, protocols
  - Support for alternatives to discontinued components
  - Support new features
How To Update?

- Define test cases for update verification
- Review non-upstream patches
  - Drop
  - Rebase
  - Rework
  - Replace

Test
Fix
Debug
How To Update?

- Define test cases for update verification
- Review non-upstream patches
  - Drop
  - Rebase
  - Rework
  - Replace
- Of course, there are real world complications
Knowledge Loss

- Developer turnover is inevitable
- Undocumented Patches will need to be ported
- Implicit assumptions *will* eventually break

---

[PATCH] optimize by dropping spinlock

Technical Debt

- Vendor forks can have thousands of patches
  - Fixes
  - Features
  - Reverts...
- Custom Interfaces to userspace especially hairy
"Soft" Vendor Lock-In

- Support for non-approved use cases can be lacking
- Patching vendor forks may be prohibitive
- Collaboration happens upstream

What if we had a clean slate?

- Choose hardware that can be supported well
  - Is there documentation?
  - Is there upstream support?
  - Is the vendor engaged upstream?

<table>
<thead>
<tr>
<th>NXP</th>
<th>TI</th>
<th>Microchip</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.MX, Layerscape</td>
<td>Sitara/AMxxxx</td>
<td>AT91</td>
</tr>
<tr>
<td>AMD/Xilinx</td>
<td>ST</td>
<td>Rockchip</td>
</tr>
<tr>
<td>ZynqMP</td>
<td>STM32MP1</td>
<td>RK3xxx</td>
</tr>
</tbody>
</table>

Examples of SoCs with good support
What if we had a clean slate?

- Prefer upstream solutions
  - Develop based on of mainline kernel instead of vendor fork
  - Avoid board-specific one-offs, reuse existing solutions
  - Collaborate with community at large, so problems and solutions are shared
Summary: Swim Upstream!

Upstream Currents

Software Package

You

Juicy Upstream Goodies
How can this look like?

- **Yocto Project**: Build upon plain Poky, reuse layers and create your own on top
- **RAUC**: Customize RAUC, a framework for fail-safe verified image-based updating
- **barebox**: Utilize unified infrastructure for atomic boot selection and more, in a kernel-like code base with a familiar UNIX-like environment
How can this look like?

- Example board: InnoComm WB15 SoM
  - NXP i.MX 8M Mini (ARM Cortex A53)
  - 8 GB eMMC, 1 GB LPDDR

How can this look like?

- Example board: InnoComm WB15 SoM
  - NXP i.MX 8M Mini (ARM Cortex A53)
  - 8 GB eMMC, 1 GB LPDDR

In the following: Guided tour through our example BSP:
https://github.com/a3f/YOCTO.BSP-ELCE2022
System Architecture

$ rauc install

system A

system B

state

barebox bootloader
System Architecture

system A

$ rauc status mark-good

system B

state

barebox bootloader
Barebox State

Reference docs: https://www.barebox.org/doc/latest/user/state.html

```c
// barebox/arch/arm/dts/imx8mm-innocomm-wb15-evk.dts
&usdhc2 { partitions {
    compatible = "fixed-partitions";
    #address-cells = <2>;
    #size-cells = <2>;
    // […]
    statepartition: partition@200000 {
        label = "barebox-state";
        reg = <0x0 0x200000 0x0 0x100000>;
    }; }
}; }

/ { state: state {
    magic = <0x11fb08ef>;
    compatible = "barebox,state";
    backend-type = "raw";
    backend = <&statepartition>;
    backend-stridesize = <0x80>;
    backend-storage-type = "direct";
    #address-cells = <1>;
    #size-cells = <1>;

    bootstate {
        system0 {
            #address-cells = <1>;
            #size-cells = <1>;
            remaining_attempts@0 {
                reg = <0x0 0x4>;
                type = "uint32";
                default = <3>;
            }
            priority@4 {
                reg = <0x4 0x4>;
                type = "uint32";
                default = <10>;
            }
        }
        // … same variables for system1
        last_chosen@10 {
            reg = <0x10 0x4>;
            type = "uint32";
        }
    }
    aliases { state = &state; }
}; }
```
// barebox/arch/arm/dts/imx8mm-innocomm-wb15-evk.dts
&usdhc2 { partitions {
    compatible = "fixed-partitions";
    #address-cells = <2>;
    #size-cells = <2>;
    // [...]
    statepartition: partition@200000 {
        label = "barebox-state";
        reg = <0x0 0x200000 0x0 0x100000>;
    }; }; }

/ { state: state {
    magic = <0x11fb08ef>;
    compatible = "barebox,state";
    backend-type = "raw";
    backend = <&statepartition>;
    backend-stridesize = <0x80>;
    backend-storage-type = "direct";
    #address-cells = <1>;
    #size-cells = <1>;
    // [...]
    state: state {
        magic = <0x11fb08ef>;
        compatible = "barebox,state";
        backend-type = "raw";
        backend = <&statepartition>;
        backend-stridesize = <0x80>;
        backend-storage-type = "direct";
        #address-cells = <1>;
        #size-cells = <1>;
    };

    #address-cells = <1>;
    #size-cells = <1>;
    remaining_attempts@0 {
        reg = <0x0 0x4>;
        type = "uint32";
        default = <3>;
    };
    priority@4 {
        reg = <0x4 0x4>;
        type = "uint32";
        default = <10>;
    };
}

// ... same variables for system1
last_chosen@10 {
    reg = <0x10 0x4>;
    type = "uint32";
}

aliases { state = &state; }

}; };

Reference docs: https://www.barebox.org/doc/latest/user/state.html
Barebox State from Userspace

- dt-utils:
  https://git.pengutronix.de/cgit/tools/dt-utils/

```bash
barebox@InnoCom:~$ vi
[...]
bootchooser.default_attempts: 4
bootchooser.reset_attempts: power-on
bootchooser.state_prefix: state.bootstate
bootchooser.boot0.boot: mmc0.root0
bootchooser.boot0.default_priority: 21
bootchooser.boot1.boot: mmc0.root1
bootchooser.boot1.default_priority: 20
bootchooser.targets: system0 system1

root@wb15-evk:~$ bootchooser --dump
bootstate.boot0.remaining_attempts=3
bootstate.boot0.priority=20
bootstate.boot1.remaining_attempts=3
bootstate.boot1.priority=30
bootstate.last_chosen=1
```
What We Need

- **Board support:**
  - Barebox
  - Linux
- **Device Tree with barebox-state node**

- **Machine layer:**
  - Image with A/B partitions
  - Barebox recipe
  - Linux recipe

- **Distro layer:**
  - RAUC recipe
  - RAUC bundle
Initial Yocto Setup

- Checkout meta layers
  - poky
  - openembedded-core
  - meta-pxt
  - meta-rauc
- source poky/oe-init-build-env
- Add layers to build/bblayer.conf
Yocto Board Support Layer

Create and add new board support layer:

- `bitbake-layer create-layer ..;/meta-wb15`
- `bitbake-layer add-layer ..;/meta-wb15`

Refer to
https://docs.yoctoproject.org/dev-manual/common-tasks.html#creating-your-own-layer
Machine Configuration

```plaintext
# meta-wb15/conf/machine/wb15.conf
DEFAULTTUNE = "cortexa53-crypto"
require conf/machine/include/arm/armv8a/tune-cortexa53.inc
SOC_FAMILY = "mx8"
include conf/machine/include/soc-family.inc
MACHINEOVERRIDES .= "::mx8m"

MACHINE_FEATURES = "usbgadget usbhost vfat ext2 serial pci touchscreen"

MACHINE_SOCARCH = "${TUNE_PKGARCH}-mx8mm"
PACKAGE_EXTRA_ARCHS:append = " ${MACHINE_SOCARCH}"

WKS_FILE = "barebox-rootfs-sd.wks.in"
```

- `DEFAULTTUNE` indirectly defined to `${DEFAULTTUNE}` via `tune-cortexa53.inc`
- `MACHINEOVERRIDES` indirectly defined to `${DEFAULTTUNE}` via `tune-cortexa53.inc`
- `MACHINE_SOCARCH` indirectly defined to `${DEFAULTTUNE}` via `tune-cortexa53.inc`
- Also sets appropriate `MACHINEOVERRIDES`
- WIC image config
Machine Configuration

```bash
# meta-wb15/conf/machine/wb15.conf (cont.)
PREFERRED_PROVIDER_virtual/kernel  := "linux-wb15"
PREFERRED_PROVIDER_virtual/dtb  := "devicetree"

PREFERRED_PROVIDER_virtual/bootloader  := "barebox"
EXTRA_IMAGEDEPENDS  += "virtual/bootloader"
BAREBOX_IMAGE = "barebox-innocomm-imx8mm-wb15-evk.img"
DDR_FIRMWARE_NAME = "lpddr4_pmu_train_1d_imem.bin [...]

MACHINE_EXTRA_RDEPENDS = "kernel-devicetree linux-firmware-imx-sdma-imx7d"
```

Reference docs:
https://docs.yoctoproject.org/dev-manual/common-tasks.html#adding-a-new-machine
Machine: Boot Firmware

- i.MX8M needs boot firmware...
  - see Rouven's talk at this ELC-E

- clone firmware-imx recipe from meta-freescale layer
  - supplies DDR training firmware
    - lpddr4_pmu_train_1d_imem.bin etc.

- add recipe for TF-A
Machine: Barebox

- .bb uses barebox.inc from meta-px
- Board support patches from mailing list
  - already merged in “next” branch :-)
  - Hint: b4 am $URL
  - Most of the board code is actually auto-generated or boilerplate... (-:)
- Default env variables
- defconfig based on barebox' imx_v8_defconfig
Machine: Barebox

```bash
# meta-wb15/recipes-bsp/barebox/barebox_2022.09.0.bb
COMPATIBLE_MACHINE = "(wb15)"

SRC_URI += "file://defconfig  file://env"
require files/patches/series.inc

BAREBOX_FIRMWARE_DIR = "${B}/firmware"
DEPENDS:append = "firmware-imx-8m trusted-firmware-a"
do_compile:prepend() {
    mkdir -p ${BAREBOX_FIRMWARE_DIR}
    cd ${DEPLOY_DIR_IMAGE}
    for fw in ${DDR_FIRMWARE_NAME}; do
        cp ${fw} ${BAREBOX_FIRMWARE_DIR}
    done
    cp bl31-imx8mm.bin ${BAREBOX_FIRMWARE_DIR}/imx8mm-bl31.bin
}
```

- copy i.MX boot firmware into Barebox build dir
- Barebox will link them into the bootloader image
# devicetree.bb

```bash
inherit devicetree

FILESEXTRAPATHS:append := \
"${THISDIR}/linux-wb15:"
SRC_URI = "\n  file://imx8mm-innocomm-wb15.dtsi \n  file://imx8mm-innocomm-wb15-evk.dts \n"

COMPATIBLE_MACHINE = "(wb15)"
```
Mainline has everything we need :-)  

defconfig:  
  - start with `make defconfig`, disable as needed
What We Need

- Board support:
  - Barebox
  - Linux
  - Device Tree with barebox-state node

- Machine layer:
  - Image with A/B partitions
  - Barebox recipe
  - Linux recipe

- Distro layer:
  - RAUC recipe
  - RAUC bundle
# meta-wb15/wic/barebox-rootfs-sd.wks.in

part barebox --source rawcopy --sourceparams="file=${BAREBOX_IMAGE},skip=1024" --
offset 1 --no-table
part / --source rootfs --fstype=ext4 --label rootA --align 1024
part / --source rootfs --fstype=ext4 --label rootB --align 1024

bootloader --ptable msdos
Distro with RAUC support

- Also see [https://github.com/rauc/meta-rauc-community](https://github.com/rauc/meta-rauc-community) for more platforms
  - but we want to build our own

```
# meta-elce2022-distro/conf/distro/elce2022.conf

require conf/distro/poky.conf
DISTRO = "elce2022"
DISTRO_VERSION = "0.0.1"

DISTRO_FEATURES += "rauc"
```
Distro: RAUC bundle

```bash
# update-bundle.bb

inherit bundle

RAUC_BUNDLE_SLOTS = "rootfs"
RAUC SLOT_rootfs = "core-image-minimal"

RAUC_KEYRING_FILE = "${THISDIR}/files/ca.cert.pem"
RAUC_CERT_FILE = "${THISDIR}/files/devel.cert.pem"

# NOTE: usually you keep the private key out of the
# Git repo, and configure it in local.conf (:-
# This is just a test key to show how it’s done
RAUC_KEY_FILE = "${THISDIR}/files/private/devel.key.pem"
```

```
meta-elce2022-distro/recipes-core/rauc

    files
        ca.cert.pem
        devel.cert.pem
        private
            devel.key.pem
        system.conf

    rauc-%.bbappend

    update-bundle.bb
```
RAUC: system.conf

```bash
# meta-elce2022-distro/recipes-core/rauc/files/system.conf
[system]
compatible=InnoCom-WB15
bootloader=barebox

[keyring]
path=/etc/rauc/ca.cert.pem

// [slot.barebox] ...

[slot.rootfs.0]
device=/dev/mmcblk0p1
type=ext4
bootname=A

[slot.rootfs.1]
device=/dev/mmcblk0p2
type=ext4
bootname=B
```

meta-elce2022-distro/recipes-core/rauc/files
---
```
ca.cert.pem
devel.cert.pem
private
devel.key.pem
system.conf
rauc_%_.bbappend
update-bundle.bb
```

use dt-utils backend
What We Need

- Board support:
  - Barebox
  - Linux
  - Device Tree with barebox-state node

- Machine layer:
  - Image with A/B partitions
  - Barebox recipe
  - Linux recipe

- Distro layer:
  - RAUC recipe
  - RAUC bundle

$ bitbake update-bundle
Thanks!

Questions?