From Zero to A/B

Swimming Upstream with

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 But deviating from upstream comes with its own costs: Vendor BSPs are often full of non-upstreamable patches, outdated components, and hardships keeping the software up-to-date.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

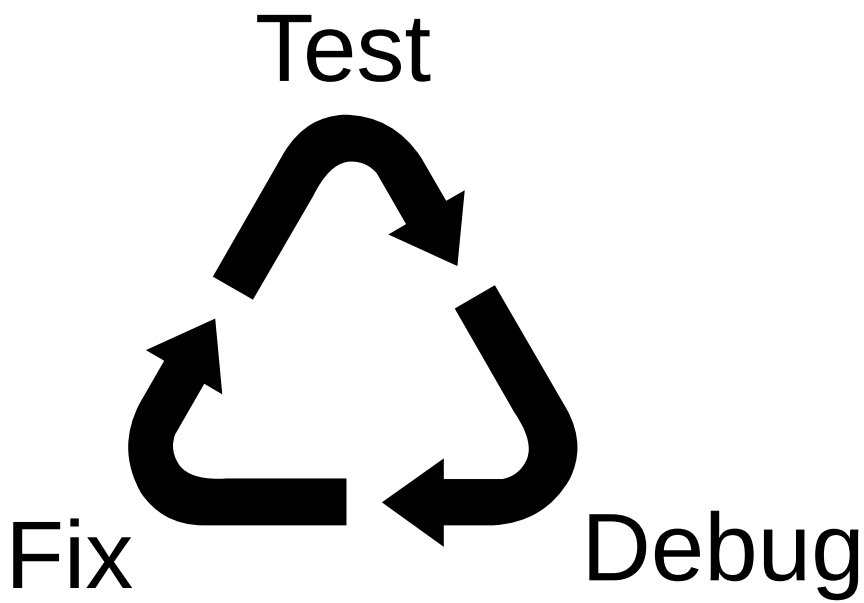
[https://tabbycatspawprints.com/2020/07/14/armchair-photo-tours-a-bear-watching-tour-at-katmai-national-park-and-preserve/]

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

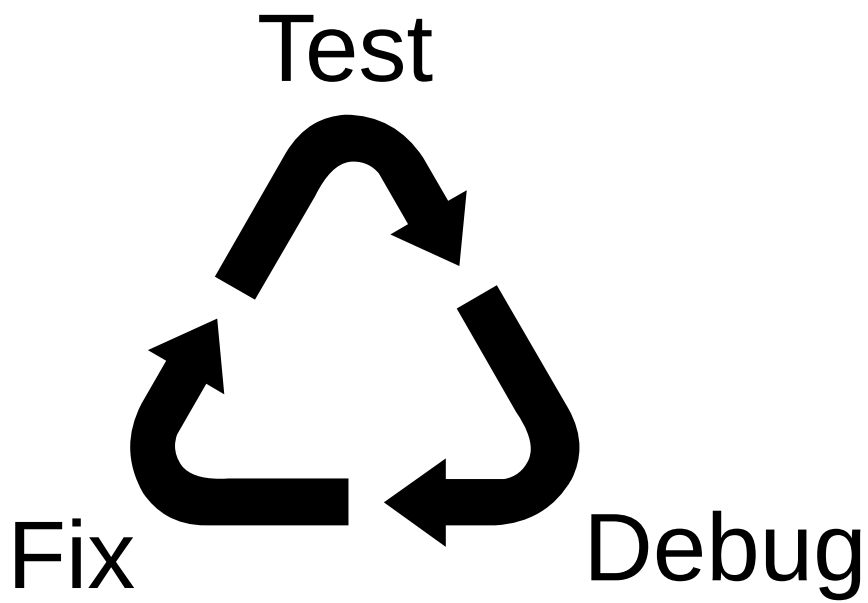
How To Update?

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



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

[http://dailynewsdig.com/pictures-of-bears-chilling-thinking-about-life/]

Technical Debt

- Vendor forks can have thousands of patches
  - Fixes
  - Features
  - Reverts…
- Custom Interfaces to userspace especially hairy

[https://www.nrtdatabase.org/trailDetail.php?recordID=2415]

“Soft” Vendor Lock-In

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

[https://www.mlive.com/news/ann-arbor/2021/08/independence-lake-beach-closed-due-to-high-e-coli-bacteria-levels.html]

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?

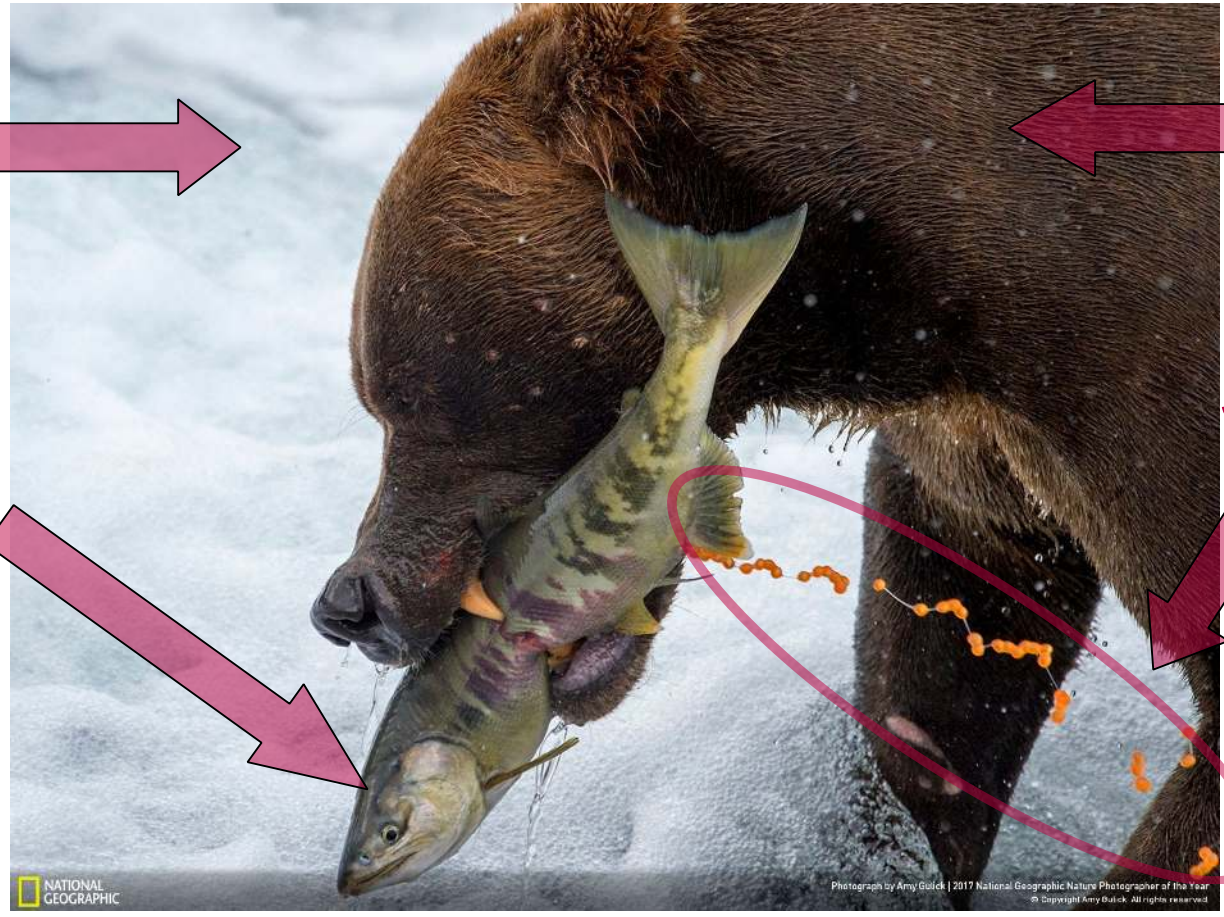
NXP i.MX, Layerscape	**TI** Sitara/AMxxxx	**Microchip** AT91
AMD/Xilinx ZynqMP	**ST** STM32MP1	**Rockchip** RK3xxx

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!

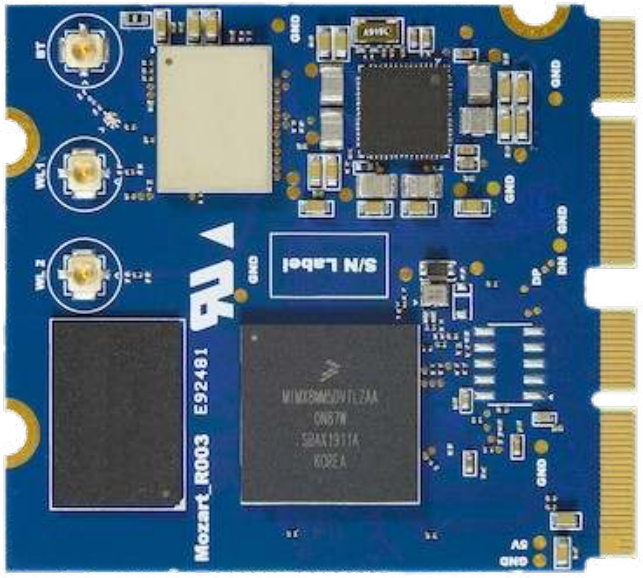


How can this look like?

- 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?

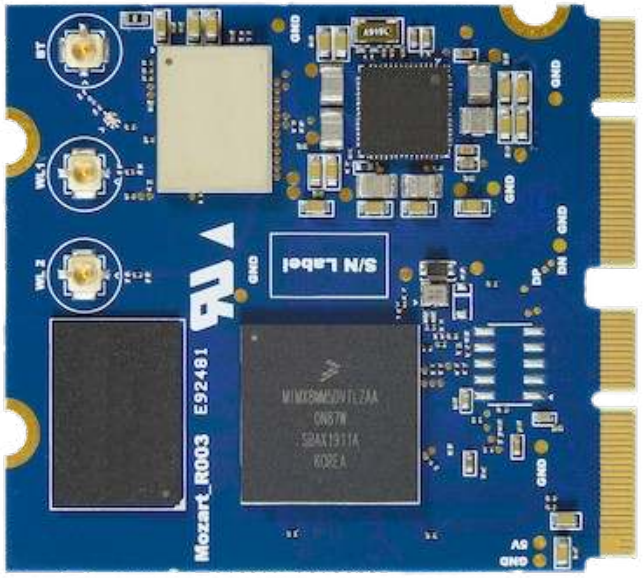
- Example board:
  InnoComm WB15 SoM
  - NXP i.MX 8M Mini (ARM Cortex A53)
  - 8 GB eMMC, 1 GB LPDDR



[https://www.innocomm.com/product_inner.aspx?num=2232]

How can this look like?

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



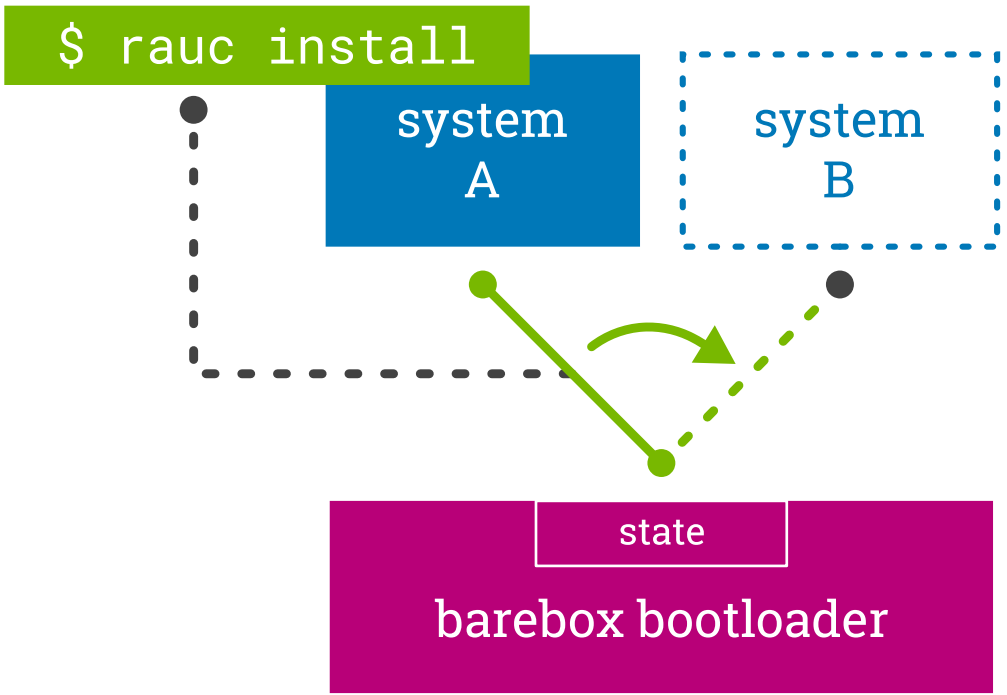
[https://www.innocomm.com/product_inner.aspx?num=2232]

In the following:

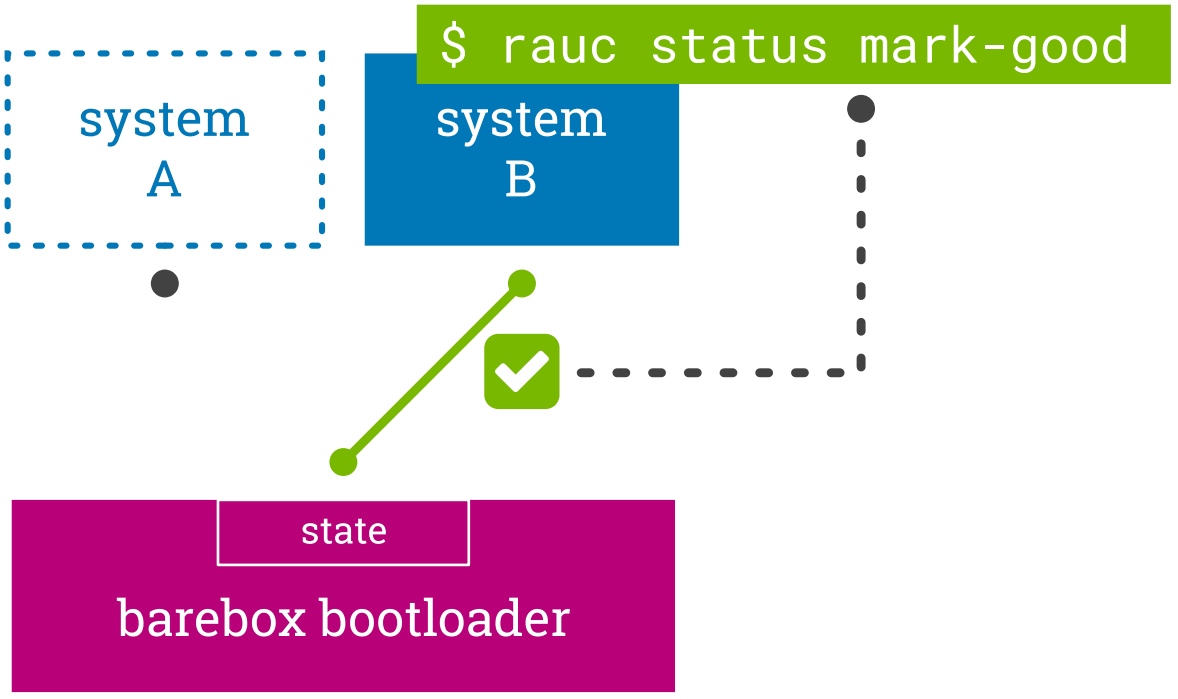
Guided tour through our example BSP:

https://github.com/a3f/YOCTO.BSP-ELCE2022

System Architecture



System Architecture



Barebox State

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

```
// 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 State

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

```
// 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 State from Userspace

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

```
barebox@InnoCom WB15-EVK:/ nv
[…]
bootchooser.default_attempts: 4
bootchooser.reset_attempts: power-on
bootchooser.state_prefix: state.bootstate
bootchooser.system0.boot: mmc0.root0
bootchooser.system0.default_priority: 21
bootchooser.system1.boot: mmc0.root1
bootchooser.system1.default_priority: 20
bootchooser.targets: system0 system1

root@wb15-evk:~ barebox-state --dump
bootstate.system0.remaining_attempts=3
bootstate.system0.priority=20
bootstate.system1.remaining_attempts=3
bootstate.system1.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-ptx
  - 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`

```
meta-wb15
├── conf
│   └── layer.conf
├── COPYING.MIT
├── README
└── recipes-example
    └── example
        └── example_0.1.bb
```

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

Machine Configuration

```
# 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"
```

also sets appropriate MACHINEOVERRIDES

indirectly defined to ${DEFAULTTUNE} via tune-cortexa53.inc

wic image config

Machine Configuration

```
# 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"
```

used by barebox recipe

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

```
meta-wb15
├── classes
│   └── fsl-eula-unpack.bbclass
├── FSL_EULA
└── recipes-bsp
    ├── firmware-imx
    │   ├── firmware-imx_8.12.bb
    │   ├── firmware-imx-8.12.inc
    │   ├── firmware-imx-8m_8.12.bb
    │   ├── firmware-imx-8m_%.bbappend
    │   └── firmware-imx_%.bbappend
    └── trusted-firmware-a
        ├── trusted-firmware-a_2.6.bb
        ├── trusted-firmware-a_%.bbappend
        └── trusted-firmware-a.inc
```

Machine: Barebox

- .bb uses barebox.inc from meta-ptx
- 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`

```
meta-wb15/recipes-bsp/barebox
├── barebox_2022.09.0.bb
├── barebox_%.bbappend
└── files
    ├── env
    │   └── nv
    │       ├── autoboot_timeout
    │       ├── boot.default
    │       ├── bootm.provide_machine_id
    │       └── [bootchooser vars…]
    ├── patches
    │   ├── 0001-add-InnoComm-WB15.patch
    │   └── series.inc
    └── wb15
        └── defconfig
```

Machine: Barebox

```
# 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

Machine: Device Tree

```
# devicetree.bb
inherit devicetree

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

COMPATIBLE_MACHINE = "(wb15)"
```

```
meta-wb15/recipes-kernel/linux
├── devicetree.bb
├── linux-wb15
│   ├── imx8mm-innocomm-wb15.dtsi
│   ├── imx8mm-innocomm-wb15-evk.dts
│   └── wb15
│       └── defconfig
├── linux-wb15_5.19.bb
└── linux-wb15.inc
```

Machine: Kernel

```
inherit kernel
LINUX_VERSION = "${PV}"
SRC_URI =
"https://www.kernel.org/pub/linux/kernel/v5.x
/linux-${LINUX_VERSION}.tar.xz"
SRC_URI += "file://defconfig"
COMPATIBLE_MACHINE = "(wb15)"
```

```
meta-wb15/recipes-kernel/linux
├── devicetree.bb
├── linux-wb15
│   ├── imx8mm-innocomm-wb15.dtsi
│   ├── imx8mm-innocomm-wb15-evk.dts
│   └── wb15
│       └── defconfig
├── linux-wb15_5.19.bb
└── linux-wb15.inc
```

- 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

Image with A/B partitioning

```
# 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 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

```
# 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

```
# 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
```

use dt-utils backend

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

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?