

One Image to Rule Them All

Portably Handling Hardware Variants



Ahmad Fatoum – a.fatoum@pengutronix.de

About Me

👤 Ahmad Fatoum

👜 Pengutronix e.K.

🐙 a3f [↗](#)

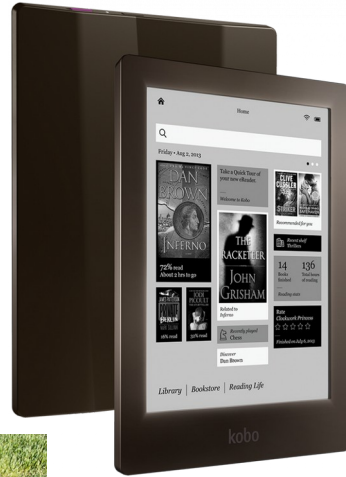
✉ a.fatoum@pengutronix.de

- Kernel and Bootloader Porting
- Driver and Graphics Development
- System Integration
- Embedded Linux Consulting



Embedded Systems

Can be very diverse



(For illustration purposes only. No affiliation)



Embedded Systems

But ...



(For illustration purposes only. No affiliation)



Embedded Systems

But sometimes less so



(For illustration purposes only. No affiliation)



Hardware Variants

- New product based on existing one
- Hardware often designed with future variants in mind
 - DNP: **D**o **N**ot **P**lace components
 - CPU meant to query hardware revision
- And if not, hardware revisions will often happen anyway
 - Try out a different chip
 - Replace unsourceable part
 - Strip down to save costs



Multiple Image vs Single Image

Multiple Images

- All differences contained in build system
- Compiler can be tuned specifically for used CPU
- Smallest Image size
- Reduced risk of breaking other platforms

Single Image

- Shorter overall build time
- Less bitrot/divergence of shared code
- Fewer artifacts
 - Easier CI and Testing
- Improved user experience:
 - Single image to flash
 - Single USB-stick to recover
 - Less pitfalls to document



Multiple Image vs Single Image

Multiple Images

- All differences contained in build system
- Compiler can be tuned specifically for used CPU
- Smallest Image size
- Reduced risk of breaking other platforms

Single Image

- Shorter overall build time
- Less bitrot/divergence of shared code
- Fewer artifacts
 - Easier CI and Testing
- Improved user experience:
 - Single image to flash
 - Single USB-stick to recover
 - Less pitfalls to document



Multiple Images

- Separate OE/Yocto MACHINE per machine
- Duplicate machine configuration with small changes
- Add MACHINE-specific overrides where needed ↗
- Images become specific to this MACHINE

```
DEPENDS:append:my-new-machine = "extra-dependency"  
do_install:append:my-new-machine() {  
    # extra steps  
}
```



Single Image

- Core idea: **shared code with dynamic configuration at runtime**
- Lots of software read config files → Need to vary configuration per hardware
- Single image per hardware ≠ Single image for all functions
 - Rule of Thumb: It should be possible to have a single command that builds all software related to a platform: Update bundles, disk images, recovery images ...
- Rest of the talk introduces a toolbox on how to achieve a single image in the order that the system boots

→ If you have questions or alternative suggestions during the talk,
just shout (or raise your hand)



In the beginning was the kernel

- Linux is our Hardware Abstraction Layer
 - Single kernel per ISA possible, but...
 - ... lots of other devices need to be matched to drivers
 - Some busses like USB and PCI are enumerable, so matching is easy
 - But most peripherals on System-on-Chips are not enumerable
- Hardware expected to come with a hardware description (ACPI/DT)



OpenFirmware Device Tree

- Data structure to describe a computer's hardware components
 - Tree of nodes with properties, e.g. `compatible = "acme,my-fancy-board"`
- Shipping Device Tree preflashed does not work well
 - Understanding of hardware evolves
 - Older kernels won't understand new bindings
 - Going against the grain: Much more coverage for matching kernel + DT
- Instead: Ship DT alongside kernel. Bootloader selects the correct one
 - [Bootloader Specification](#)[↗][↗], [distroboot](#)[↗], [FIT](#)[↗], [UKI](#)[↗]..



Pitfalls with FIT

- Hardcoded load and entry addresses
 - Bootloader should figure it out
- Bootloader hardcodes configuration name
 - → Bootloader should compare DT compatibles [↗](#)
- OE-core's kernel-fitimage.bbclass does both these things by default...



Bootloader

- Needs to figure out what hardware it's running on
- Let's assume for now: bootloader outside main storage area
 - Arguably still *One Image to rule them all*
 - Thanks to eMMC HW boot partitions [↗](#), increasingly common configuration
- Different bootloader per board
 - Native support: barebox multi images [↗](#), TF-A for STM32MP15 [↗](#)
 - BSP-side support: Yocto UBOOT_CONFIG [↗](#)
 - Yocto multi-config [↗](#) (Overkill for this use case)



Device-Tree Fixups

- Appropriate kernel device tree has been selected
- Fixups code is run on it to manipulate it:
 - Default: `/chosen/bootargs`, `/chosen/linux,initrd-{start,end}`
 - Extra: `mac-address`, `/memory` nodes
 - Board-specific and architecture specific fixups possible



Device-Tree Fixups: Board-specific

- Detect available hardware somehow and fix it up 

```
barebox@Sandbox:/ of_property -fs display0 compatible powertip,ph320240t028_zha  
barebox@Sandbox:/ of_diff /mnt/tftp/afa-oftree-${global.hostname} +
```

```
panel {  
-     compatible = "powertip,ph320240t023_iha";  
+     compatible = "powertip,ph320240t028_zha";  
};
```



Device-Tree Fixups continued

- Board-specific fixups [↗](#)[↗](#)

```
static int my_board_display_fixup(struct device_node *root, void *_data)
{
    struct device_node *display = of_find_node_by_alias(root, "display0");
    if (!display)
        return -EINVAL;

    return of_property_write_string(display, "compatible",
                                    "powertip,ph320240t028_zha");
}

static int my_customboard_probe(struct device *dev)
{
    return of_register_fixup(my_board_display_fixup, NULL);
}
```

- For U-Boot, don't forget `fdt_increase_size()`! [↗](#)



Device-Tree Fixups continued

- arch-specific fixups. U-Boot example[↗]:

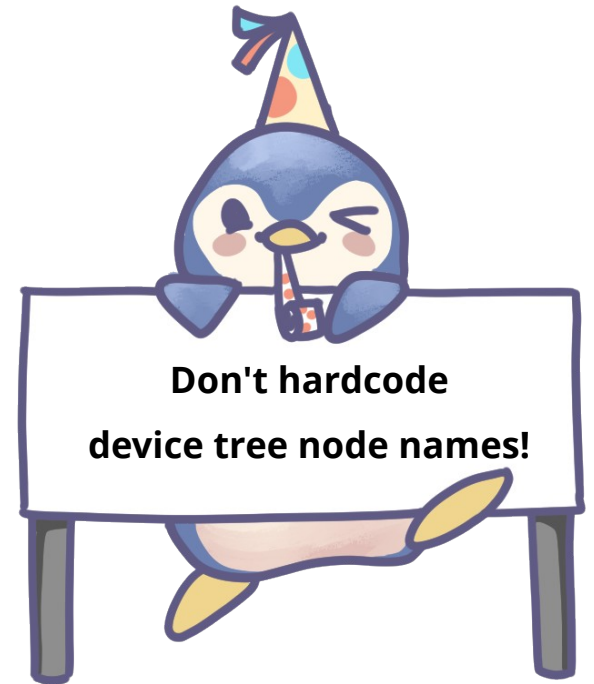
```
int disable_vpu_nodes(void *blob)
{
    static const char * const nodes_path_8mm[] = {
        "/vpu_g1@38300000", "/vpu_g2@38310000",
    };

    if (is_imx8mm_lite())
        return disable_fdt_nodes(blob,
            nodes_path_8mm,
            ARRAY_SIZE(nodes_path_8mm));

    return -EPERM;
}

int ft_system_setup(void *blob, struct bd_info *bd)
{
    disable_vpu_nodes(blob);
}
```

- Look up by alias[↗] or compatible[↗] where possible.
- In barebox use `of_get_node_by_reproducible_name`[↗]



Device-Tree Overlays

- Useful for optional daughter boards / capes
- Special case of device tree fixup, changes described as DT
- Supported by BLSpec, distroboot & FIT
- Caveat: can't delete nodes! But can override the status property

```
/dts-v1/;  
/plugin/  
  
&i2c0 {  
    rtc@68 {  
        compatible = "dallas,ds3231";  
        reg = <0x68>;  
    };  
};
```



DT Overlays in Linux

- Patch set available for many years
 - Exists in weird limbo state of being partially merged
 - A lot of complexity most projects can do without
 - Drivers need awareness that DT changed under them
 - `of_mutex` held during overlay application
- Just let the bootloader handle it
(or interface directly with FPGA manager if required)

Off to Userspace

Kernel got a DT
What now?



Making applications portable

- User application will need to interact with the real world
- Everything is a file symlink
- Use existing symlinks, don't hardcode device hierarchies

```
- /sys/devices/platform/soc@0/soc@0:bus@30000000/30350000.ocotp-ctrl/imx-ocotp0/nvmem  
+ /sys/bus/nvmem/devices/imx-ocotp0/nvmem
```



- Or create new ones as appropriate

```
ACTION=="add", SUBSYSTEM=="tty", KERNEL=="ttymxc2", SYMLINK="ttySTM"
```

- Added benefit: More robust across kernel updates



Block device paths

- `/dev/mmcblkXpY`
 - Stable since v5.10 if DT aliases are used 
 - Good for kernel root=, but different variants can have different partitioning
- `/dev/sdX`
 - No guarantees. Need udev rules for stable symlinks



udev: block device paths

udev ships with `60-persistent-storage.rules`, but: *last processed device wins*

- `/dev/disk/by-uuid`, `/dev/disk/by-label`
 - File System/Partition UUID may be the same for A/B partitions in A/B system
- `/dev/disk/by-partlabel`, `/dev/disk/by-partuuid`
 - User inserts both SD with same image as eMMC
- `/dev/disk/by-id`
 - Factors in device serial, which makes it cumbersome for using same image
- `/dev/disk/by-path`
 - Upstream rename: `platform-03b60000.sdhci` → `platform-3b60000.mmc`



udev: stable block device paths

- Instead: Write your own rules, identifying devices by usage or by stable topology

```
ENV{ID_PART_ENTRY_SCHEME}=="gpt", \  
ENV{ID_PART_ENTRY_NAME}=="?*", ENV{ID_PATH}=="?*", \  
SYMLINK+="disk/by-pathlabel/${env{ID_PATH}/${env{ID_PART_ENTRY_NAME}}"
```

- Result: /dev/disk/by-pathlabel/platform-30b60000.mmc/rootfs-a →
../../../../mmcb1k0p3
- Hopefully coming soon to a systemd-udev near you (systemd PR #29219) [↗](#)



udev: /sys symlinks via DT Aliases

- Not how you're supposed to use it, but easy way to get stable links into /sys

```
/ { aliases { eeprom0 = &eeprom_som; } }
```

```
ACTION=="add", ENV{OF_ALIAS_0}=="?* ", RUN+="/bin/mkdir -p /dev/by-ofalias", \  
  RUN+="/bin/ln -sf /sys%p /dev/by-ofalias/%E{OF_ALIAS_0}"
```

```
$ ls /dev/by-ofalias/  
eeprom0/   gpio0/     gpio2/     i2c1/      mmc0/      sai1/  
serial1/   serial3/   ethernet0/ gpio1/     serial2/   spi0/
```

```
readlink /dev/by-ofalias/eeprom0  
/sys/devices/platform/soc@0/30800000.bus/30a50000.i2c/i2c-1/1-0050/1-00501
```



udev: matching by compatible [↗](#)

```
#!/bin/sh
# save as /usr/lib/udev/of_base_compatible
printf 'OF_BASE_COMPATIBLE="%s"\n' "$(tr '\0' ' ' < \
    /sys/firmware/devicetree/base/compatible)"
```

```
ACTION=="remove", GOTO="system_partitions_end"
SUBSYSTEM!="block", GOTO="system_partitions_end"
```

```
IMPORT{program}="of_base_compatible"
```

```
ENV{OF_BASE_COMPATIBLE}=="*acme,quirky-board*", GOTO=quirky_board
```



systemd: Condition*

- udev created symlinks can be used with `ConditionPathExists`[↗]
- Fallback: Match against board compatible with `ConditionFirmware`

[Unit]

Description=Read/Write Storage, persistent over Reboots and Updates

run before services causing high CPU load

Before=systemd-udev.service systemd-journald.service systemd-udev-trigger.service

ConditionFirmware=device-tree-compatible(acme,fastest-booting-device)

[Mount]

use symlink to data partition outside of /dev to get rid of udev dependency

What=/dev/mmcblk0p4

Where=/data

Type=ext4

Options=rw,nosuid,noexec

systemd: use well-known targets

- `Required-by=boot-complete.target`
- `Wanted-by=graphical.target`

`[Unit]`

`Before=graphical.target`

`Requires=weston.socket`

`[Service]`

`Type=notify`

`EnvironmentFile=/etc/default/weston`

`ExecStart=/usr/bin/weston --modules=systemd-notify.so`

`[Install]`

`WantedBy=graphical.target`



systemd-networkd

- Matching by device tree compatible possible since v251

```
[Match]
Name=swp3
KernelCommandLine=!nfsroot
Firmware=device-tree-compatible(acme,best-switch)

[Network]
Bridge=br0

[BridgeVLAN]
PVID=1
EgressUntagged=1
VLAN=2
```



Runtime configuration generation

```
#!/bin/sh

original=/etc/xdg/weston/weston.ini

if [ -e "/data/weston.ini" ]; then
    ln -s -f /data/weston.ini "$1/weston.ini"
    exit 0
fi

ini="$(mktemp -p "$1" weston.ini.XXXXXX)"

{
    echo "# originally from '$original'";
    cat $original;
    echo "# automatically generated from XXXX";
    echo;
} >>"$ini"

# [ insert dynamic stuff here ]

mv "$ini" "$1/weston.ini"
```

```
[Unit]
Description="Weston, a Wayland compositor"
RequiresMountsFor=/run
RequiresMountsFor=/data

[Service]
Environment="XDG_CONFIG_HOME=/run"
ExecStartPre=weston-config-write.sh /run
ExecStart=weston
```



Back to the Bootloader



RAUC Variants: Configuration [↗](#)

```
[system]
compatible=acme-imx8
bootloader=barebox
statusfile=/data/rauc-statusfile
variant-file=/sys/devices/soc0/soc_id
data-directory=/data/rauc

[slot.bootloader.0]
device=/dev/mmcblk0
type=boot-emmc

[handlers]
post-install=/usr/bin/rauc-post-install
```

```
inherit bundle
```

```
[...]
```

```
RAUC_SLOT_barebox-imx8mm = "barebox"
RAUC_SLOT_barebox-imx8mm[type] = "boot"
RAUC_SLOT_barebox-imx8mm[file] = "barebox-acme-imx8mm.img"
# No .i.MX8MM suffix to stay backwards compatible
RAUC_SLOT_barebox-imx8mm[name] = "bootloader"

RAUC_SLOT_barebox-imx8mn = "barebox"
RAUC_SLOT_barebox-imx8mn[type] = "boot"
RAUC_SLOT_barebox-imx8mn[file] = "barebox-acme-imx8mn.img"
RAUC_SLOT_barebox-imx8mn[name] = "bootloader.i.MX8MN"
RAUC_SLOT_barebox-imx8mn[offset] = "-32K"

BOOT_SLOTS += "barebox-imx8mm barebox-imx8mn"
```



RAUC Variants: Bundle Manifest

```
$ rauc info acme-prod-bundle.raucb
[...]
4 Images:
  [rootfs]
    Filename:  acme-imx8m.squashfs-xz.verity.img
    Checksum:  2a59d59e3809f827ce709d3815e3950eef4a6a93af5557a93a7fdffa71460843
    Size:      100.1 MB (219123712 bytes)
    Adaptive:  block-hash-index
  [boot-files]
    Filename:  signed-fitImage-acme-verity-setup-imx8m-imx8m.img
    Checksum:  fa51fd49abf67705d6a35d18218c115ff5633aec1f9ebfdc9d5d4956416f57f6
    Size:      20.7 MB (20725494 bytes)
  [bootloader]
    Filename:  barebox-acme-imx8mm.img
    Checksum:  9a3058157de8b004fc5ddeea90813a3bba456c76dfa4b9c6dc0dcc64476d8f8d
    Size:      1.0 MB (1045032 bytes)
  [bootloader]
    Variant:   i.MX8MN
    Filename:  barebox-acme-imx8mn.img
    Checksum:  60e6aa4ad2d315ff8ab59a827637d123fdb7af4107f97c9344a1863a59568aca
    Size:      1.0 MB (1006760 bytes)
```



Going a step further

Can we have a single bootloader?



Single Bootloader: Why bother?

- Useful when many variants need to be supported
 - Easier factory bootstrap
 - Single USB-Stick or SD to recover
 - Smaller update bundle size
 - No confusion about what's the correct bootloader image
 - QSPI, recovery, 2x2Gbit RAM, Variant C with a side of fries

Bootloader: support multiple boards

- Target common subset of all boards
 - Completely different SoCs: Binary acrobatics can make different entry points work
 - Similar enough SoCs:
 - U-Boot: Mostly infeasible, because build is for single SoC due to `__weak` and `#ifdef`
 - barebox: probes completely from device tree on a number of platforms (`CONFIG_ARCH_MULTIARCH`)
- Detect Board type
 - Read EEPROM, probe I2C devices, sample strapping pins, check fusebox, ... etc.
 - Set own DT compatible according to detected board type
- Recommended: Manipulate bootloader device tree.
 - Otherwise, you lose out on the differences: e.g. no network boot, USB recovery

Bootloader: dynamic device tree

- Use separate device trees in prebootloader [↗](#)

```
extern char __dtb_z_imx8mn_evk_start[], __dtb_z_imx8mn_ddr4_evk_start[];
void *fdt;

/* Check if we configured DDR4 in EL3 */
if (readl(MX8M_DDRC_CTL_BASE_ADDR) & BIT(4))
    fdt = __dtb_z_imx8mn_ddr4_evk_start;
else
    fdt = __dtb_z_imx8mn_evk_start;

imx8mn_barebox_entry(fdt);
```

- Apply a fixup onto the device tree [↗](#)

```
- of_register_fixup(my_board_switch_fixup, &variant);
+ my_board_switch_fixup(of_get_root_node(), &variant);
```



Bootloader: dynamic device tree

- Apply a built-in overlay [↗](#)

```
obj-$(CONFIG_OVERLAY_LIVE) += acme_best_switch.dtbo.o
```

```
int acme_probe(struct device *dev)
{
    /* [ ... ] */
    overlay = of_unflatten_dtb(match_data, INT_MAX);
    of_overlay_apply_tree(dev->of_node, overlay);
    /* [ ... ] */
}

extern char __dtbo_acme_best_switch_start[];

static const struct of_device_id acme_board_of_match[] = {
    { .compatible = "acme,best-switch", .data = __dtbo_acme_best_switch_start },
};
```



Summary

- Bootloader
 - Probe board type, fixup kernel DT and possibly bootloader's own DT
- Linux
 - Receive hardware appropriate device tree
- systemd
 - Activate units only if system fullfills conditions
- User software
 - select configuration depending on board
 - generate configuration at runtime
 - use portable symlinks, with board-specific destination
 - beware the hardcoded number: GPIO sysfs indices, IRQ numbers, disk order, ... etc.
- But they must be consistently used, otherwise death by thousand papercuts
 - The best time to do so is when adding the first platform

