barebox Bells and Whistles

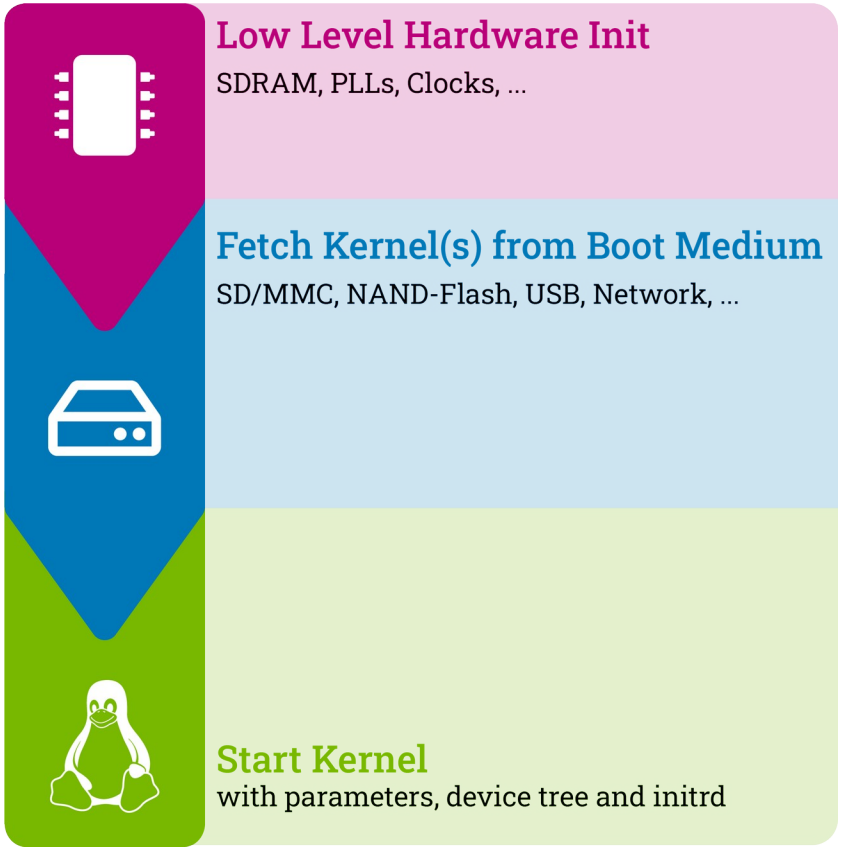
Beyond „Just“ Booting

Ahmad Fatoum – a.fatoum@pengutronix.de

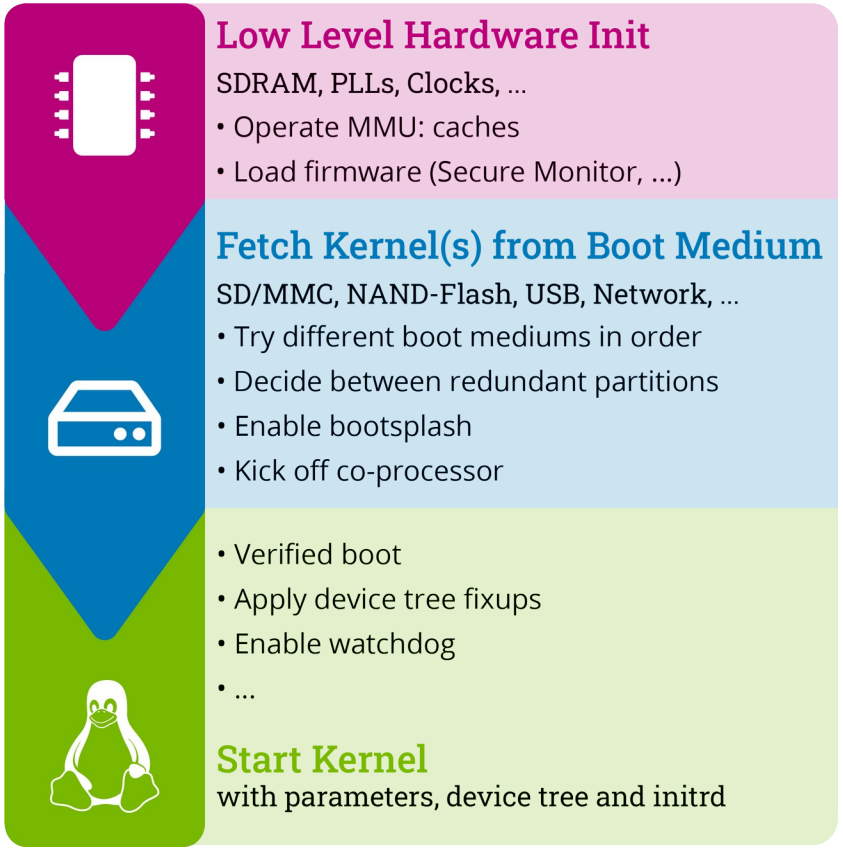
Agenda

- What is barebox
- Porting barebox to a new board
- Customization
- Booting with barebox
- Bring-Up
- Recent Developments

Bootloader: bare minimum



Bootloader: modern expectation



Bootloader: Scalability and Maintainability

- This gets complex real fast, so one could appreciate
  - Driver Model, Separate Hardware Description (Device Tree)
  - Abstractions: Virtual File System (VFS), block layer, character devices
  - Interactive Prompt for debugging
  - Scriptability for ease of use
  - Persistent environment for development
  - Introspection (Peek/Poke, bus transfers) for bring up

Bootloader: Scalability and Maintainability

- This gets complex real fast, so one could appreciate
  - ✔ Driver Model, Separate Hardware Description (Device Tree)
  - ✔ Abstractions: Virtual File System (VFS), block layer, character devices
  - ✔ Interactive Prompt for debugging
  - ✔ Scriptability for ease of use
  - ✔ Persistent environment for development
  - ✔ Introspection (Peek/Poke, bus transfers) for bring up
- Linux with built-in initramfs could check all the boxes, but
  - ✗ bootloader is often size constrained (e.g. 64K SRAM)

Bootloader: barebox

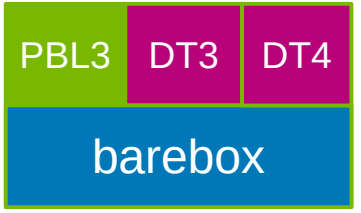
- Started as U-Boot-v2 in 2007
- Renamed to barebox in 2009
- GPL-2.0 licensed
- Monthly Release Cycle
- Supports ARM, MIPS, x86 EFI, RISC-V
- Linux-like driver API, coding style, Kconfig, Kbuild
- POSIX-like user API
- UNIX-like interactive shell

barebox: First and Second-Stage

- Traditionally boot loader (BL) is split into:
  - First Stage (FSBL) : Does low-level initialization, loads Second-Stage (BIOS on PC)
  - Second-Stage (SSBL): Runs from SDRAM, Loads the Kernel (OS bootloader)
- barebox can be used as both, but on some SoCs only second-stage is implemented

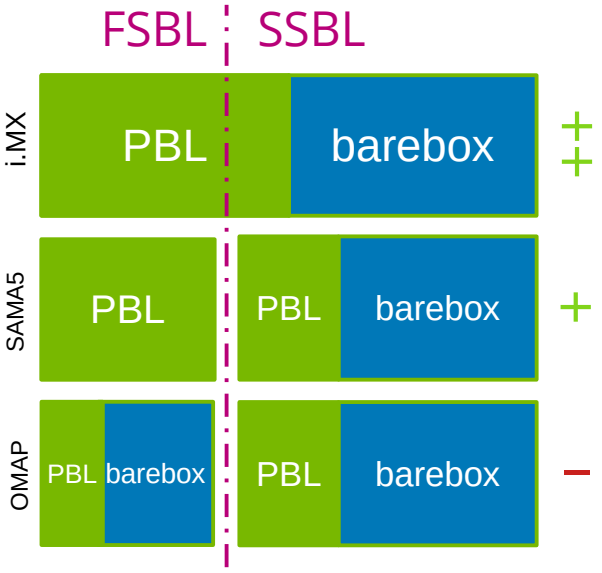
barebox: second-stage bootloader (SSBL)

- Linux multi-platform kernel is great, we want that in the bootloader as well!
  But: kernel is passed device tree blob (DTB) from bootloader, we are the bootloader
  - board-agnostic barebox is passed a DTB
  - For each enabled board: link (compressed) barebox proper with board-specific pre-bootloader (PBL) that passes a DTB



barebox: first-stage bootloader (FSBL)

- Set up clocks and SDRAM in PBL, load full image from boot medium directly into SDRAM
  - Good for Development: imx_v7_defconfig builds images for 119 boards at once
  - Good for Integration: One barebox recipe, one Kconfig
- Alternative: Build barebox twice, override FSBL init to chainload SSBL



Examples of barebox SD image structure

barebox: Porting Your Board

➔ **Add device tree**

- Add new PBL entry point
- Add board driver if necessary
- Tell Kconfig and Kbuild about them
- Register new multi-image entry point

- barebox regularly imports all Linux device tree sources into `/dts/src`
- barebox DTS usually imports the upstream DTS and extends it as necessary

DT

```
--- /dev/null
+++ b/arch/arm/dts/stm32mp157c-odyssey.dts
+#include <arm/stm32mp157c-odyssey.dts>
+#include "stm32mp151.dtsi"
+
+/ {
+  chosen {
+    environment-emmc {
+        compatible = "barebox,environment";
+        device-path = &sdmmc2, "partname:barebox-environment";
+    };
+  };
+};
+
+&phy0 {
+  reset-gpios = <&gpiog 0 GPIO_ACTIVE_LOW>;
+};
```

barebox: Porting Your Board

- Add device tree
- ➔ **Add new PBL entry point**
- Add board driver if necessary
- Tell Kconfig and Kbuild about them
- Register new multi-image entry point

- First barebox code run after header
- Does Low Level Initialization (Stack, caches, ...)
- Print a „>“ if `CONFIG_DEBUG_LL=y`
- Call barebox entry with device tree and memory base/size


```
--- /dev/null
+++ b/arch/arm/boards/seeed-odyssey/lowlevel.c
+#include <common.h>
+#include <mach/entry.h>
+#include <debug_ll.h>
+
+extern char __dtb_z_stm32mp157c_odyssey_start[];
+
+ENTRY_FUNCTION(start_stm32mp157c_seeed_odyssey, r0, r1, r2)
+{
+   void *fdt;
+
+   stm32mp_cpu_lowlevel_init();
+
+   putc_ll('>');
+
+   fdt = __dtb_z_stm32mp157c_odyssey_start + get_runtime_offset();
+
+   stm32mp1_barebox_entry(fdt);
+}
```

barebox: Porting Your Board

- Add device tree
- Add new PBL entry point
- ➔ **Add board driver if necessary**
- Tell Kconfig and Kbuild about them
- Register new multi-image entry point

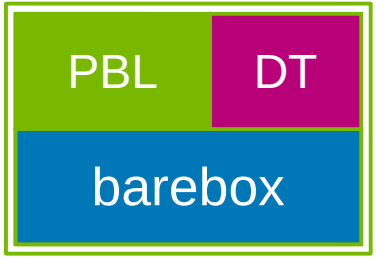
- Handle Hardware Quirks
- Register Device Tree fixups for boot
- Modify (unflattened) built-in DT

```
--- /dev/null
+++ b/arch/arm/boards/seeed-odyssey/board.c
+#include <init.h>
+#include <mach/bbu.h>
+#include <driver.h>
+
+static int odyssey_som_probe(struct device_d *dev)
+{
+  barebox_set_model("Odyssey");
+  return stm32mp_bbu_mmc_register_handler("emmc", "/dev/mmc1.ssbl", BBU_HANDLER_FLAG_DEFAULT);
+}
+static const struct of_device_id odyssey_of_match[] = {
+       { .compatible = "seeed,stm32mp157c-odyssey-som" },
+       { /* sentinel */ },
+};
+static struct driver_d odyssey_som_driver = {
+       .name = "odyssey-som",
+       .probe = odyssey_som_probe,
+       .of_compatible = odyssey_of_match,
+};
+device_platform_driver(odyssey_som_driver);
```

barebox: Porting Your Board

- Add device tree
- Add new PBL entry point
- Add board driver if necessary
- ➔ **Tell Kconfig and Kbuild about them**
- Register new multi-image entry point

  - `obj-y` for normal normal objects
  - `lwl-y` for low-level (here PBL) objects
  - `bbenv-y` for Environment



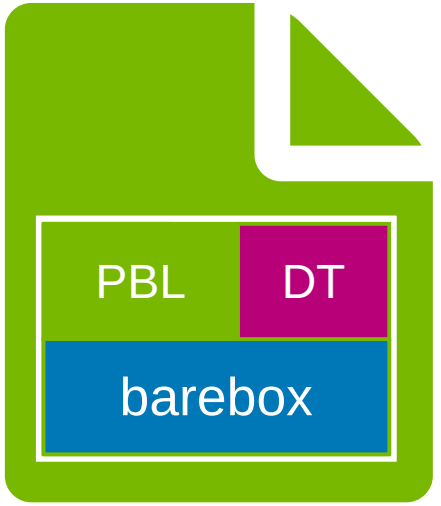

```
--- /dev/null
+++ b/arch/arm/boards/seeed-odyssey/Makefile
+lwl-y += lowlevel.o
+obj-y += board.o

--- a/arch/arm/mach-stm32mp/Kconfig
+++ b/arch/arm/mach-stm32mp/Kconfig
+config MACH_SEEED_ODYSSEY
+   select ARCH_STM32MP157
+   bool "Seeed Studio Odyssey"

--- a/arch/arm/boards/Makefile
+++ b/arch/arm/boards/Makefile
+obj-$(CONFIG_MACH_SEEED_ODYSSEY)       += seeed-odyssey/

--- a/arch/arm/dts/Makefile
+++ b/arch/arm/dts/Makefile
+lwl-$(CONFIG_MACH_SEEED_ODYSSEY) += stm32mp157c-odyssey.dtb.o
```

barebox: Porting Your Board

- Add Device Tree
- Add new PBL entry point
- Add board driver if necessary
- Tell Kconfig and Kbuild about them

➔ **Register new multi-image entry point**

- Add new entry point to list of PBLs for the multi-image build
- Name the new image
- Specify format to use (here .stm32)




```
--- a/images/Makefile.stm32mp
+++ b/images/Makefile.stm32mp
$(obj)/%.stm32: $(obj)/% FORCE
   $(call if_changed,stm32_image)

STM32MP1_OPTS = -a 0xc0100000 -e 0xc0100000 -v1

+pblb-$(CONFIG_MACH_SEEED_ODYSSEY) += start_stm32mp157c_seeed_odyssey
+FILE_barebox-stm32mp157c-seeed-odyssey.img =
start_stm32mp157c_seeed_odyssey.pblb.stm32
+OPTS_start_stm32mp157c_seeed_odyssey.pblb.stm32 = $(STM32MP1_OPTS)
+image-$(CONFIG_MACH_SEEED_ODYSSEY) += barebox-stm32mp157c-seeed-
odyssey.img
```

barebox: Porting Your Board

```
barebox 2020.09.0 #1099 Mon Sep 28 21:05:54 CEST 2020


Board: Seeed Studio Odyssey-STM32MP157C Board
STM32 RCC reset reason RST (MP_RSTSR: 0x00000054)
stm32mp-init: detected STM32MP157CAC Rev.B
stpmic1-i2c stpmic10: PMIC Chip Version: 0x10
psci psci.of: detected version 1.1
mdio_bus: miibus0: probed
eth0: got preset MAC address: 00:80:e1:42:52:29
stm32_sdmmc 58005000.sdmmc@58005000.of: registered as mmc0
stm32_sdmmc 58007000.sdmmc@58007000.of: registered as mmc1
mmc1: detected MMC card version 5.0
mmc1: registered mmc1.boot0
mmc1: registered mmc1.boot1
mmc1: registered mmc1
stm32-iwdg 5a002000.watchdog@5a002000.of: probed
remoteproc0: 10000000.m4@10000000.of is available
netconsole: registered as netconsole-1
malloc space: 0xcfe7e700 -> 0xdfcfcdff (size 254.5 MiB)
envfs: no envfs (magic mismatch) - envfs never written?

Hit m for menu or any to stop autoboot:    0
Booting entry 'net'
eth0: 1000Mbps full duplex link detected
T eth0: DHCP client bound to address 172.17.2.88
could not open /mnt/tftp/none-linux-stm32mp157c-odyssey: No such file or directory
ERROR: Booting entry 'net' failed
Nothing bootable found
barebox@Odyssey:/
```

- SSBL can now be flashed and run
- barebox now tries to net boot

Customization: The Environment

- Compile-time configuration via the built-in environment
  - default, feature-specific, board-specific and external (BSP) environments are overlaid
  - Kconfig is not the place for board-specific configuration (breaks multi-image)
- Board Code and init scripts can read/write the runtime environment
- Can be persisted for development

```
barebox@Odyssey:/ nv autoboot=abort user=afa
nv variable modified, will save nv variables on shutdown
barebox@Odyssey:/ ls -R /env
/env:
bin        boot       data       init       nv
/env/bin:
mtdparts-add
/env/boot:
bnet    net
/env/data:
ansi-colors      boot-template
/env/init:
automount         automount-ratp    ps1
/env/nv:
allow_color         autoboot            autoboot_timeout
user
```

```
defaultenv/defaultenv-2-base/
├── bin
│   └── mtdparts-add
├── boot
│   ├── bnet
│   └── net
├── data
│   ├── ansi-colors
│   └── boot-template
├── init
│   ├── automount
│   ├── automount-ratp
│   └── ps1
└── nv
    ├── allow_color
    ├── autoboot_timeout
    └── user
```

Customization: Magic Variables

- Interaction with barebox core happens via magic variables
- Most are `global.variables`, these are initialized from the correspoding non-volatile `nv.variables` on startup

```
barebox@STM32MP157C-DK2:/ magicvar
OPTARG                          optarg for hush builtin getopt
PS1                             hush prompt
automount_path                  mountpath passed to automount scripts
bootsource                      The source barebox has been booted from
global.autoboot_timeout         Timeout before autoboot starts in seconds
global.boot.default             default boot order
global.boot.watchdog_timeout    Watchdog enable timeout in seconds before booting
global.bootm.provide_machine_id If true, add systemd.machine_id= with value global.machine_id to Kernel
global.hostname                 shortname of the board. Also used as hostname for DHCP requests
global.linux.bootargs.*         Linux bootargs variables
global.linux.bootargs.console   console= argument for Linux from the stdout-path property in /chosen node
…
```

Customization: Device Parameters

- Driver runtime configuration happens via device parameters
- Accesses can call back into the driver
- Can have types beside strings and be read-only

```
dev_add_param_enum(dev, "next",
        reboot_mode_param_set, NULL,
        &reboot->reboot_mode_next,
        reboot->modes,
        reboot->nmodes, reboot);
```

```
barebox@STM32MP157C-DK2:/ devinfo tamp.reboot_mode
Driver: syscon-reboot-mode
Bus: platform
Parent: 5c00a000.tamp@5c00a000.of
Parameters:
  next: normal (type: enum) (values: "normal", "loader", "fastboot")
  prev: normal (type: enum) (values: "normal", "loader", "fastboot")
Device node: /soc/tamp@5c00a000/reboot-mode
reboot-mode {
        compatible = "syscon-reboot-mode";
        offset = <0x150>;
        mask = <0xff>;
        mode-normal = <0x0>;
        mode-loader = <0xbb>;
        mode-fastboot = <0xfb>;
};
barebox@STM32MP157C-DK2:/ echo $tamp.reboot_mode.prev
normal
barebox@STM32MP157C-DK2:/ tamp.reboot_mode.next=fastboot
barebox@STM32MP157C-DK2:/ reset -r stm32-rcc
```

Customization: Hush Scripts

- Custom commands can be added to `/env/bin`
- Core shell utilities allow interacting with the character devices in the VFS
- default barebox init sources some scripts on startup
  - `/env/init/*`, `/env/bmode/$global.reboot_mode.prev`

```
barebox@Linux Automation MC-1 board:/ edit /env/boot/mmc
```

```
 /env/boot/mmc     <ctrl-d>: Save and quit <ctrl-c>: quit
#!/bin/sh

if [ -n "$nv.boot.default" ]; then
     exit
fi

if [ $bootsource = mmc ]; then
     global.boot.default="mmc${bootsource_instance}.root net"
fi
~
```

```
barebox@Linux Automation MC-1 board:/ ls /dev
eeprom0                         full
hwrng0                          mdio0-phy03
mem                             mmc1
mmc1.0                          mmc1.1
mmc1.2                          mmc1.3
mmc1.barebox-environment        mmc1.boot0
mmc1.boot1                      mmc1.data
mmc1.root                       mmc1.ssbl
netconsole-1                    null
prng                            ram0
remoteproc0                     serial0-1
stm32-bsec                      stpmic10
zero
```

Putting this all together: Booting

```
barebox@STM32MP157C-DK2:/ help boot

boot - boot from script, device, ...

Usage: boot [-vdlmwt] [BOOTSRC...]

This is for booting based on scripts. Unlike the bootm command which can boot a single image
this command offers the possibility to boot with scripts (by default placed under /env/boot/).

BOOTSRC can be:
- a filename under /env/boot/
- a full path to a boot script
- a device name
- a partition name under /dev/
- a full path to a directory which
   - contains boot scripts, or
   - contains a loader/entries/ directory containing bootspec entries

Multiple bootsources may be given which are probed in order until one succeeds.

Options:
   -v Increase verbosity
   -d Dryrun. See what happens but do no actually boot
   -l List available boot sources
   -m Show a menu with boot options
   -w SECS   Start watchdog with timeout SECS before booting
   -t SECS   specify timeout in SECS
```

Booting: Nice to Have

- The partition itself should define how it wants to be booted
  - kernel, device tree, initrd, dt-overlays and boot arguments all in one place
- Detect everything else
  - Memory Layout? Allocate correctly aligned non-overlapping images buffers automatically
  - File System? Automount detected file system on first access
  - Image Format? Call appropriate handler for detected type
  - Don’t repeat yourself: bootloader could already determine correct `root=` and `console=`
- Boot parameters still exactly specifiable if needed

Booting: Bootloader Specification

```
barebox@STM32MP157C-DK2:/ cat /mnt/mmc0.4/loader/entries/stm32mp157c-dk2.conf
title            PTXdist - Pengutronix-DistroKit
version          5.9
options          rootwait rw
linux            /boot/zImage
devicetree       /boot/stm32mp157c-dk2.dtb
linux-appendroot true
barebox@STM32MP157C-DK2:/ boot -d mmc0.4
blspec: ignoring entry with incompatible devicetree "atmel,sama5d27-som1-ek"
Booting entry 'PTXdist - Pengutronix-DistroKit'
blspec: booting PTXdist - Pengutronix-DistroKit from mmc0
Adding "root=PARTUUID=11b06d74-571f-4c00-9bf4-0d6ea769f433" to Kernel commandline

Loading ARM Linux zImage '/mnt/mmc0.4//boot/zImage'
Loading devicetree from '/mnt/mmc0.4//boot/stm32mp157c-dk2.dtb'
commandline: root=PARTUUID=11b06d74-571f-4c00-9bf4-0d6ea769f433 console=ttySTM0,115200n8 rootwait rw
Dryrun. Aborted
```

- `linux-appendroot` is a barebox extension for appending the correct `root=`
  - Bootspec can be storage agnostic (Export over NFS and boot it all the same)
- `echo mmc0.4 > $BSP/barebox/env/nv/boot.default` and you're good to go

Booting: Bootchooser

```
barebox@STM32MP157C-DK2:/ bootchooser -i
Good targets (first will be booted next):
system1
    id: 2
    priority: 20
    default_priority: 20
    remaining attempts: 3
    default attempts: 3
    boot: 'mmc0.root1'
system0
    id: 1
    priority: 10
    default_priority: 21
    remaining attempts: 2
    default attempts: 3
    boot: 'mmc0.root0'

Disabled targets:
none

last booted target: system1
```

- Having two rootfs partitions allows flashing from running system and fallback after failed update
- Need to detect update failure:
  - Watchdog triggers
  - User application doesn‘t mark boot as good
- Need mutable variable storage
  - Environment is inadequate
    - Lacks redundancy and atomicity
    - Nice to have: authorization, wear leveling, access control
  - Solution: barebox-state

Booting: barebox-state

```
barebox@STM32MP157C-DK2:/ of_dump /state
state {
     magic = <0x4b434d63>;
        compatible ="barebox,state";
        backend-type = "raw";
        backend = <0x87>;
        backend-stridesize = <0x40>;
        backend-storage-type = "direct";
        bootstate {
                #address-cells = <0x1>;
                #size-cells = <0x1>;
                system0 {
                        #address-cells = <0x1>;
                        #size-cells = <0x1>;
                        remaining_attempts@0 {
                                reg = <0x0 0x4>;
                                type = "uint32";
                                default = <0x4>;
                        };
                        priority@4 {
                                reg = <0x4 0x4>;
                                type = "uint32";
                                default = <0x15>;
                        };
                };
                system1 {
                        /* [snip] same vars as system 0 */
                };
                last_chosen@10 {
                        reg = <0x10 0x4>;
                        type = "uint32";
                };
        };
};
```

- State described in barebox device tree and fixed up into kernel's
- Backend can be any device barebox knows how to write to
- Userspace `barebox-state` utility for OS access
  - dt-utils to parse DTB, udev to find correct Linux device
  - No further configuration needed
- Maintains three copies for redundancy and atomicity
- CRC32 for corruption detection
- Optional HMAC for detecting unauthorized changes
- Optional Wear Leveling
- Strictly for variable storage. No need for mutable bbenv in the field
  - One less thing to worry about!
- Works great with RAUC (Embedded Recipes Talk)

Booting: Final /env/nv

```
barebox@STM32MP157C-DK2:/  nv
  autoboot_timeout: 0
  boot.default: bootchooser
  boot.watchdog_timeout: 20
  bootchooser.default_attempts: 4
  bootchooser.reset_attempts: power-on
  bootchooser.retry: 1
  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
  bootm.provide_machine_id: 1
  linux.bootargs.loglevel: loglevel=5 systemd.log_level=warning
systemd.show_status=auto
```

- Not a single script needed, all is configuration :-)

Boot: Tim Toady

- `boot` collects boot entries and calls the handler if requested
- If you have special requirements, consider configuring `bootm` directly

```
barebox@STM32MP157C-DK2:/ bootm /mnt/tftp/${global.user}-barebox-${global.hostname}"
barebox@STM32MP157C-DK2:/ automount -l
/mnt/tftp            ifup -a && mount -t tftp $global.net.server /mnt/tftp
```

- Multiple `bootm` handlers (uImage, FIT, ELF, ...) and new ones can be easily added

```
static struct image_handler image_handler_stm32_image_v1_handler = {
    .name = "STM32 image (v1)",
    .bootm = do_bootm_stm32image,
    .filetype = filetype_stm32_image_v1,
};

static int stm32mp_register_stm32image_image_handler(void)
{
    return register_image_handler(&image_handler_stm32_image_v1_handler);
}
late_initcall(stm32mp_register_stm32image_image_handler);
```

Bring-Up: Drivers

- Most Subsystem APIs imported from Linux. Often, drivers are fairly easy to port
- But no interrupts in barebox: Sometimes porting from other bootloaders easier, but needs special attention to multi-image incompatibilities
  - `#ifdef`s outside headers
  - `__attribute__((weak))`
  - clashing `#define`s
  - Configuration hardcoded globally at compile-time
- Or roll your own, you can port your barebox driver into the kernel later

Bring-Up: Commands

- Shell has access to character devices for I/O memory, EEPROMs, block device partitions, OTP, (net-)consoles, regmaps, PHYs … etc.
- Commands for device tree manipulation
  - Example: `echo -o /env/init/fix-sd-card-detect of_property mmc0 -s -f broken-cd ; saveenv`
- Useful defaults (`bnet`, `net`, `/mnt/nfs`) for network boot
  - `nv user=you` is often all you need for net booting an unconfigured board
  - If network is unavailable, ratpfs allows mounting host directories over a serial link
- Help text can be compiled in, Documentation mechanically extracted into Sphinx docs

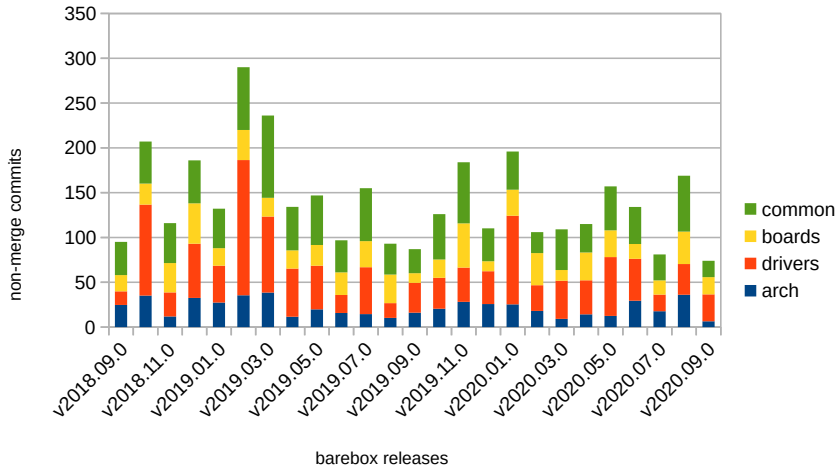
Bring-Up: Custom Commands

- barebox „userspace“ is a POSIX-like programming environment
  - Userspace code is easier to port
- Kernel-API also available: Control GPIOs, handle SPI transfers, ...

```
#include <common.h>
#include <command.h>
#include <complete.h>

static int do_true(int argc, char *argv[])
{
    return 0;
}

static const char * const true_aliases[] = { ":", NULL};

BAREBOX_CMD_START(true)
    .aliases  = true_aliases,
    .cmd      = do_true,
    BAREBOX_CMD_DESC("do nothing, successfully")
    BAREBOX_CMD_GROUP(CMD_GRP_SCRIPT)
    BAREBOX_CMD_COMPLETE(empty_complete)
BAREBOX_CMD_END
```

Recent Developments (in the last two years)

- Architecture Support:
  - i.MX8M[M/Q/P]
  - RISC-V
  - Kalray MPPA
  - ARM64 Layerscape
  - STM32MP1
  - Raspberry Pi 3
- ubootvarfs
- OP-TEE (Early-)Loading
- AddressSanitizer (arm32, arm64 and sandbox), UBSan, CONFIG_COMPILE_TEST
- Device tree Overlays
- Deep Probe
- Workqueues and Slices



Interested?

- Project home: https://barebox.org
- Collaboration via Mailing List

  https://lists.infradead.org/mailman/listinfo/barebox
- #barebox on Freenode
- Give `ARCH=sandbox` a whirl!

```
user@host$ git clone https://git.pengutronix.de/git/barebox
user@host$ cd barebox
user@host$ make sandbox_defconfig
user@host$ make
user@host$ ./barebox
```

Thanks!

Questions?