Recent Advances in U-Boot

Simon Glass, Google LLC
@imonglass

Thursday 29 June 2023
Agenda

● What is U-Boot?
● Complexity in firmware
● How U-Boot helps with complexity
● New things in U-Boot in the last few years
● Demo
U-Boot

● Universal boot loader
  ○ Boot anything on anything
  ○ Project has been running for about 20 years
  ○ Typically 6k commits each year; under very active development
  ○ Four releases each year; release candidate every two weeks

● Large feature set
  ○ Around 3m lines of C code; some tools are in Python
  ○ Kernel style, shares APIs with Linux, configured with Kconfig
  ○ Main architectures are ARM, PowerPC, RISC-V, x86
  ○ CI covers a large subset of features

● On the forefront of embedded firmware technology
Why is U-Boot so popular?

- Supports most of the features that people want in a bootloader
  - Large array of board support
  - Linux compatibility / easy porting

- Easy to modify and extend
  - Relatively simple code base
  - Most feature development can be done on the host (sandbox builds)
  - Single-threaded (no locking infrastructure or concurrency problems)
  - Good documentation and test infrastructure

- Open to new ideas and features

- Consistent release schedule
The challenge of increasing complexity

- Complexity is growing in many areas
  - SoCs - more IP blocks, power domains, multiple CPU types
  - Firmware packaging - private tools and techniques
  - Security / signing - SoC-specific with many variations
  - Boot flow / firmware fragmentation - multiple firmware projects in one firmware image
  - Build and device configuration - different product models, features enabled/disabled

- What is U-Boot doing to cope with this complexity?
Dealing with SoC complexity

- U-Boot's driver model provides
  - Linux-compatible devicetree support
  - Over 100 driver classes, e.g. BLK, MMC, PCI, VIDEO
  - Parent / child relationships and automatic private data
  - Relatively easy porting from Linux (e.g. MTD layer)

```c
%i2c0 {
    clock-frequency = <4000000>;
    i2c-scl-rising-time-ns = <168>;
    i2c-scl-falling-time-ns = <4>;
    status = "okay";

    rk808: pmic@1b {
        compatible = "rockchip,rk808";
        reg = <0x1b>;
        interrupts-parent = <&gpio3>;
        interrupts = <10 IRQ_TYPE_LEVEL_LOW>;
        #clock-cells = <1>;
        clock-output-names = "xin32k", "rk808-clkout2";
        pinctrl-names = "default";
        pinctrl-0 = <&pmic_int_l>;
        rockchip.system-power-controller;
        wakeup-source;
    }
}
```
Complexity example: pinctrl, clocks, power

- Automatic pinmux, clock, power domains
- To get the first MMC device:
  - `uclass_get_device(UCLASS_MMC, 0)`
- U-Boot selects the pin muxing, enables required power domains and clocks

```c
static int mmc_power_init(struct mmc *mmc) {
    ...
    ret = device_get_supply_regulator(mmc->dev, "vmmc-supply", &mmc->vmmc_supply);
};

&sdmmc {
    bus-width = <4>;
    cap-sd-highspeed;
    cd-gpios = &gpio0 7 GPIO_ACTIVE_LOW;
    disable-wp;
    max-frequency = <150000000>;
    pinctrl-names = "default";
    pinctrl-0 = &sdmmc_clk &sdmmc_cmd &sdmmc_bus;
    vmmc-supply = <&vcc3v0_sd>;
    vqmmc-supply = <&vcc_sdio>;
    status = "okay";
};
```
Complexity example 2: Configuration

- Problem: many similar models based on a common design
- Traditional solution: one build for each model
- Better solution: run-time configuration
  - Single U-Boot build for all models
  - Devicetree describes the hardware
- U-Boot handles the differences at runtime
  - Devices instantiated based on devicetree
  - Device parameters come from devicetree
- Pass configuration between firmware components

```c
fifo-depth = <256>;
priv->fifo_depth = dev_read_u32_default(dev, "fifo-depth", 0);
```
Complexity 3: Firmware Packaging

- 'Binman' tool collects binaries into an image
  - Binaries come from build systems
  - Image is the final firmware loaded into the device

- Data-driven operation, using an image description
  - Models an image as an ordered list of entries
  - Each has properties such as offset, size, contents, alignment, compression
  - Binman loads the input files, puts them together, writes the output image(s)
  - Works in parallel, typically in a single pass, so is extremely fast
  - Very easy to modify the image as needed; allows use of CONFIG options and entry arguments
  - Tool dependencies are included in the description
  - Supports FIT, FIP, CBFS, IWFI, etc.

- Provides a way to build and fetch vendor tools

```plaintext
rom {
  filename = "u-boot.rom";
  size = <0x40000000>;
  pad-byte = <0xff>;
  mkimage {
    args = "-n rk3399 -T rkspi";
    u-boot-spl {
    };
  }
  u-boot {
    offset = <0x300000>;
  }
  u-boot-img {
    offset = <0x400000>;
  }
  fdtmap {
  }
}
```
Standard boot

● U-Boot has always had powerful booting features
  ○ Flat Image Tree (FIT) for multiple images (kernel, ramdisk, FPGA)
  ○ Signature verification, compression
  ○ Load a collection of images based on "vendor,model" compatible strings

● Standard boot adds a higher-level interface
  ○ Automatically locate boot devices
  ○ Automatically search for distros to boot
  ○ Provide a menu of available options

● Replaces 'run distro_bootcmd'
  ○ Easier configuration (generally none at all)
Standard boot - unifying all boot methods

● Three basic concepts
  ○ `bootdev` - storage devices to be scanned
  ○ `bootflow` - an OS to boot
  ○ `bootmeth` - methods for finding bootflows on bootdevs

● `bootdev` and `bootmeth` are uclasses
  ○ We have drivers for MMC, USB
  ○ Bootmeth drivers for syslinux, EFI, ChromiumOS, custom

● `bootflow` is simply a data structure
  ○ E.g. points to a extlinux.conf file, a .efi executable
  ○ May not have a file at all
  ○ Indicates which bootmeth to use to boot

● U-Boot scans for available bootflows, provides a menu for the user
UEFI support

- EFI_LOADER provides a UEFI layer in U-Boot
  - Full GPL implementation supports booting distros like Ubuntu, Fedora
  - Supports UEFI secure boot; provides capsule updates, TPM measurement
  - Makes use of existing U-Boot drivers, so generally there is no need to adjust board support
  - Includes a boot-manager implementation along with menu support

- U-Boot can also run as an EFI application

* Future
  Complete full boot/update support including ARM FWU

```bash
-> bootflow scan -lb
Scanning for bootflows in all bootdevs
Seq Method       State   Uclass    Part  Name                      Filename
---  -----------  ------  --------  ----  ------------------------  ----------------
Scanning global bootmeth 'efi_mgr':
Hunting with: nvme
Hunting with: qfw
Hunting with: scsi
scanning bus for devices...
Hunting with: virtio
Scanning bootdev 'qfw_pio.bootdev':
  fatal: no kernel available
Scanning bootdev 'virtio-blk#0.bootdev':
  0  efi          ready   virtio       1  virtio-blk#0.bootdev.part efi/boot/bootx64.efi
** Booting bootflow 'virtio-blk#0.bootdev.part_1' with efi
EFI using ACPI tables at f0060
  efi_install_fdt() WARNING: Can't have ACPI table and device tree - ignoring DT.
  efi_run_image() Booting /efi\boot\bootx64.efi
```
VBE - Verified Boot for Embedded

- A true UEFI alternative
- Scope
  - boot flow
  - image selection
  - update
- Uses FIT to package firmware / OS images
- Uses fwupd to perform firmware update
- You know in advance what you are booting and what it needs
  - No EFI callbacks
- See osfc'22 talk: 'Introduction to VBE Verified Boot for Embedded'

* Future
Future: A/B firmware update sample implementation on RockPRO64
Documentation

- U-Boot moved to rST a few years ago
  - Uses Sphinx, following Linux's lead
  - 'make htmldocs' builds the documentation
  - Allows patches to include documentation updates
  - Supports deep links, images, etc.
- Most existing documentation has been converted
  - Currently around 80K lines of rST
  - Some 80 commands (out of ~250) are documented
  - Some existing features are still undocumented, or not rST

* Future
  Complete documentation for all commands and features
Testing and CI

- Expanded significantly over the past few years
  - Uses gitlab infrastructure with ~6 runners
  - Each run takes approx. 70 minutes to complete
  - Local tests can run in a few minutes (e.g. 'make pcheck')
  - Around 1120 tests in total
- Sandbox + emulators for fast, easy tests

*Future*
- Easier distributed labs with Labgrid
- Code-coverage tracking
- Booting common distros in CI

See Demo
Devicetree and Schema

● U-Boot has used devicetree since 2011 (same year as Linux)
  ○ As a result there are quite a few differences in bindings
  ○ These are being resolved SoC by SoC
  ○ U-Boot has some schema 'upstream' (bootph-xxx and options/ node)

● Use of livetree (hierarchical data structure) is expanding
  ○ Provides an easy way to access nodes: ofnode
  ○ Provides an easy way for devices to read properties: dev_read...(dev, "prop")
  ○ Faster for updates; multiple trees are now supported
  ○ Some work on moving devicetree fix-ups to ofnode

* Future
Move schema upstream; run schema validation on U-Boot tree
Quality-of-life improvements

- **Kconfig migration**
  - Completed as of 2023.01
  - Very large effort by many people, over ~6 years
  - Provides a path to drop board-specific config.h files

- **Text-based environment**
  - Simple syntax in a text file
  - Avoids use of #defines in config.h files

- **Link-time Optimisation (LTO)**

- **U-Boot shows a logo!**

- **Events**
  - Allows 'spying' on events such as new-device creation
  - Alternative to weak functions, with better visibility and auditing (event-dumper tool)

---

```c
// SPDX-License-Identifier: GPL-2.0+
/*
 * Copyright (c) Siemens AG, 2023
 * Authors:
 *   Jan Kiszka <jan.kiszka@siemens.com>
 */

usb_pgood_delay=900
watchdog_timeout_ms=CONFIG_WATCHDOG_TIMEOUT_MSECS
start_watchdog=
  if test ${watchdog_timeout_ms} -gt 0; then
    wdt dev watchdog@40610000;
    wdt start ${watchdog_timeout_ms};
    echo Watchdog started, timeout ${watchdog_timeout_ms};
  fi
```

* Future
  - Updated HUSH shell
Cross-project communication

- Firmware Handoff (bloblist in U-Boot)
  - Provides a way to pass tagged data from one project to another
  - E.g. U-Boot can pass memory information to/from TF-A, OP-TEE
  - [github.com/FirmwareHandoff](https://github.com/FirmwareHandoff)

* Future
  Industry-wide, universal format for firmware images
Networking

- TCP/IP support and wget
- IPv6
- New PHY API

* Future
  Discussions about moving to lwip
RISC-V and x86

- **RISC-V boards now up to 21**
  - Boards from AndesTech, SiFive, Microchip, OpenPiton, Sipeed
  - Running in CI with QEMU
- **Booting distros supported on x86 (pending patches)**
- **Coreboot support has been enhanced**
  - Uses SPCR to find UART
  - `cbsysinfo` command shows the sysinfo table
  - Now runs in CI with QEMU
Tracing

- Used (with bootstage) to find bottlenecks in boot
- Record function entry / exit
- Export data for use with trace-cmd and kernelshark
- Also supports an interactive flamegraph

See Demo

'Cyclic' subsystem

- Provides a way to run things in the background
  - Register a function to be called, setting a period in microseconds
  - The function will be called when U-Boot is idle

- Many possible (future) uses
  - Resetting the watchdog timer (implemented in 2022.10)
  - Scanning the USB bus in the background
  - Read files from the network in the background
  - Scanning for bootflows in the background

GUI and menus

- New 'expo' subsystem supports graphical / text display
  - Arranged as a series of 'scenes', each with a list of items to display
  - The user can move through scenes using the keyboard
  - So far the only supported items are menus
- New 'cedit' command allows the user to edit configurations*
  - Like the BIOS configuration machine on x86 devices

* patches pending

```yaml
scenes {
    main {
        id = <ID_SCENE1>;
        title-id = <ID_SCENE1_TITLE>;
        prompt = "UP and DOWN to choose, ENTER to select";
        cpu-speed {
            type = "menu";
            id = <ID_CPU_SPEED>;
            title = "CPU speed";
            title-id = <ID_CPU_SPEED_TITLE>;
            item-label = "2 GHz", "2.5 GHz", "3 GHz";
            item-id = <ID_CPU_SPEED_1 ID_CPU_SPEED_2 ID_CPU_SPEED_3>;
        };
        power-loss {
            type = "menu";
            id = <ID_POWER_LOSS>;
            title = "AC Power";
            item-label = "Always Off", "Always On", "Memory";
        };
    };
}
```

* Future
Load / save configuration
Demo

- Standard boot
- Binman
- CI
- Tracing
- Configuration editor
Thank you for listening

- U-Boot is an open-source firmware project
- Patches and ideas are welcome

- My details
  - Simon Glass
  - to: u-boot@lists.denx.de
  - cc: sjg@chromium.org