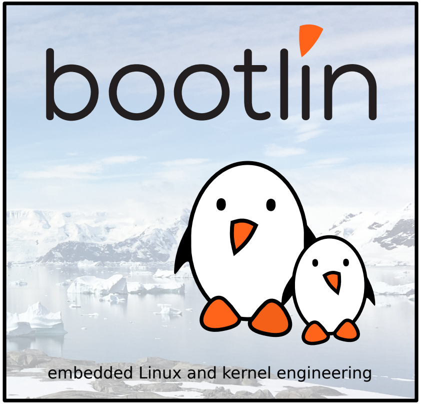
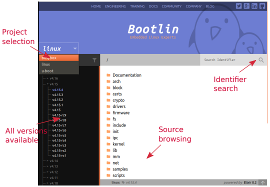
Implementing A/B System Updates with U-Boot

Michael Opdenacker
michael.opdenacker@bootlin.com



Michael Opdenacker

- Founder and Embedded Linux engineer at Bootlin:
  - Embedded Linux **expertise**
  - **Development**, consulting and training
  - Strong open-source focus
- About myself:
  - Always happy to learn from every new project, and share what I learn.
  - Initial author of Bootlin's freely available embedded Linux, kernel and boot time reduction training materials (https://bootlin.com/docs/)
  - Current documentation maintainer for the Yocto Project
  - Current maintainer of the Elixir Cross Referencer, making it easier to study the sources of big C projects like the Linux kernel. See https://elixir.bootlin.com.



Abstract

A popular way to implement system updates is through the A/B scheme, in which you have two copies of the root filesystem, one which is active, and one that is meant to contain the next update. When a new update is successfully applied, you need to make the corresponding partition become the new active one. That's when a number of practical questions arise, such as how to identify the active partition, how to detect when the new system fails to boot properly, and how to fall back to the previous version?

It was hard to find documentation about how U-Boot could address such needs to implement a functional and failsafe A/B system update mechanism. This presentation proposes to address this need by sharing the practical solutions we found, using lesser known commands and capabilities in U-Boot. We will also explain how the Linux side can cooperate with the U-Boot side. Fortunately, you won't need to erase half of your brain to get updated on this topic.

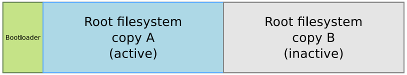
Introduction

Context and experience

- Had the opportunity to explore system update techniques, to prepare slides and labs for a customer seminar (not ready to publish in its current form).
- We also wanted to let participants practise with the *A/B update technique*, using the *swupdate* solution.
- Though details about the theory and how to apply the updates were easy to find, some practical aspects of switching between between versions needed more research.
- This presentation aims at sharing our findings and our practical experience.

Full image update approaches

- Consists in deploying to devices a new complete image of the system.
- Simplest and safest approach: everything is updated at the same time → guarantees that all components of the system are consistent and well tested.
- Not possible to update the system while it is running. → two partionning approaches are commonly used:
  1. *A/B scheme*: two full copies of the root filesystem, alternating between active/inactive
  2. *Rescue system*: one full copy of the system, and one smaller rescue system used to do the update
- Bootloader integration to switch between systems.



A/B scheme

Rescue system scheme

Full image update: typical scenario

With *A/B scheme*:

1. The system runs from active copy A
2. An update is triggered, either by bringing an update image locally (SD card, USB stick) or over the network
3. The update image is flashed/written on copy B, currently inactive
4. If the flashing is successful, the bootloader configuration is updated to mark copy B as the active copy
5. The system is rebooted, and boots on the new system installed in copy B.
6. Watchdog used to detect a non-functional system, and fallback to the old copy if necessary.

Main practical questions

- How to identify the active partition?
  - From U-Boot to boot the right version of the system
  - From Linux to deploy the upgrade in the non-active partition.
- How to swap the active and inactive partitions?
  - From Linux after flashing an upgrade
  - From U-Boot after failing to boot a new version of the system.
- How to fall back to the original version if the update fails to boot properly?

Here, we will address these questions in a generic way, without using bootloader integration implemented by software update solutions (*swupdate*, *RAUC*, *Mender*...)

RAUC

MENDER

Switching between root filesystems

Using partition bootable flags

When booting on block storage, a solution to specify the "active" root partition is to add a `bootable` flag to it.

- This should be done after an update is applied successfully.
- The command to run in Linux will depend on the type of partition table.
  - Using a GPT partition table:
    - You can use the `sgdisk` command to toggle the `bootable` status (bit attribute `2`) of specified partitions, for example:
      `sgdisk -A 4:toggle:2 -A 5:toggle:2 /dev/mmcblk0`
    - This assumes that only one of the two partitions has the `bootable` flag. This command makes it lose it and the other one get it.
  - Using an MBR (legacy) partition table:
    - You can use the `parted` command to add the `bootable` flag to **only one partition**:
      `parted -s /dev/mmcblk0 set 3 boot on`
    - Note that `sgdisk` cannot be used here, because it would convert your MBR partition table into a GPT one, which may not be supported by the rom code of your SoC.

Detecting the active partition in Linux

You can use the below commands to detect the current bootable partition from Linux, and deduce on which partition the next update should be applied.

- Using a GPT partition table, for example:
  `sgdisk -A 4:get:2 /dev/sdc | cut -d: -f3`
  This gets the bootable status (bit attribute `2`) of partition 4.
  If it is set, you will get `1`, otherwise `0`.
- Using an MBR partition table, for example:
  `BOOTABLE=$(parted /dev/mmcblk0 print | grep boot | cut -d' ' -f2)`
  This returns the number of the partition which is marked as bootable (assuming there is only one).

Detecting the active partition in U-Boot at boot time

Here's a solution to detect the bootable partition U-Boot

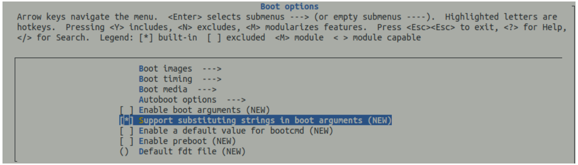
- Compile U-Boot with support for the `part` command (`CONFIG_CMD_PART`).
- You can then run:
  `part list mmc 0 -bootable bootpart`
- In the `bootpart` environment variable, you will have the number of the bootable partition on the first MMC device.
- You can then use such information to load the Linux kernel and DTB from the corresponding partition, and to set the Linux kernel command line accordingly.

U-Boot: booting on the active partition (1)

A first solution is to have a variable part in `bootargs`

- You need to compile U-Boot with `CONFIG_BOOTARGS_SUBST`
- Then you can use variables in `bootargs`:
  `setenv bootargs 'console=ttyS0 root=/dev/mmcblk0p${bootpart}'`
- And of course set the boot command to load the kernel and DTB from the right partition:

```
=> setenv bootcmd 'part list mmc 0 -bootable bootpart;
   load mmc 0:${bootpart} 21000000 boot/zImage;
   load mmc 0:${bootpart} 21000000 boot/dtb;
   bootz 21000000 - 22000000'
```



U-Boot: booting on the active partition (2)

Another solution is to use U-Boot's support for *extlinux*

- You need to compile U-Boot with `CONFIG_CMD_SYSBOOT`
- This way, U-Boot can load the kernel, DTB, initramfs and kernel command line from a specification in an `extlinux.conf` file.
- This works for booting on local storage (USB, MMC, etc) but also on the network (PXE and DHCP).
- This also allows for multiple boot options, such as a recovery filesystem non-default option.

See https://u-boot.readthedocs.io/en/latest/develop/distro.html for details.

```
Hit any key to stop autoboot:  0
Boot over mmc1!
switch to partitions #0, OK
mmc1(part 0) is current device
Scanning mmc 1:2...
Found /mmc1_extlinux/stm32mp157a-proto_extlinux.conf
Retrieving file: /mmc1_extlinux/stm32mp157a-proto_extlinux.conf
896 bytes read in 34 ms (25.4 KiB/s)
Retrieving file: /splash.bmp
18244 bytes read in 34 ms (523.4 KiB/s)
Select the boot mode
1:      OpenSTLinux
2:      stm32mp157a-proto
3:      stm32mp157a-proto-mipi050
4:      stm32mp157a-proto-rgb070
Enter choice:
```

U-Boot: using extlinux support (1)

- You need a configuration file, for example `boot/extlinux/extlinux.conf`:

  ```
  label buildroot-sysupdate
  kernel /boot/zImage
  devicetree /boot/at91-sama5d3_xplained.dtb
  append root=/dev/mmcblk0p${bootpart} rootwait console=ttyS0
  ```

- As environment variables can be used, you don't need to tweak this file for each possible root partition.

U-Boot: using extlinux support (2)

- Several environment variables need to be set:
  - `kernel_addr_r`: address in RAM to load the kernel image
  - `ramdisk_addr_r`: address in RAM to load the initramfs image (if any)
  - `fdt_addr_r`: address in RAM to load the DTB (Flattened Device Tree)
  - `pxefile_addr_r`: address in RAM to load the configuration file (usually `extlinux.conf`)
  - `bootfile`: the path to the configuration file, for example `/boot/extlinux/extlinux.conf`
- You can now load the files and boot Linux, for example:

```
=> setenv bootcmd 'part list mmc 0 -bootable bootpart;
   sysboot mmc 0:${bootpart} any'
```

  `any`: works with any filesystem supported by U-Boot.

- Note: according to U-Boot documentation, U-Boot should automatically read `extlinux.conf` from the partition with the `bootable` flag. It didn't seem to work on Microchip SAMA5D3 Xplained. PEBKAC issue?

Recovering from a failed upgrade

Recovering from a failed upgrade - Challenges

Assuming an A/B partitioning scheme

- The case when the update fails to apply is easy to manage. This is easy to detect and you stick to the current root filesystem.
- In case of success, you also need to confirm that the new root filesystem also works correctly, and otherwise fall back to the older root partition.

Assuming a single root filesystem and a rescue filesystem

- If the update fails, the new system may not boot at all.
- You then also need a mechanism in the bootloader to fall back to the rescue filesystem, and wait for a new update.

U-Boot bootcount and bootlimit (1)

U-Boot implements a mechanism to detect fail boot attempts

- Enable this through `CONFIG_BOOTCOUNT_LIMIT`
- After an update is applied, set the U-Boot environment variables `upgrade_available` to `1`, and `bootcount` to `0`.
- When U-Boot starts, if `upgrade_available` is different from `0`, `bootcount` is incremented and saved.
- If `bootlimit` is defined and `bootcount > bootlimit`, U-Boot will run the commands in `altbootcmd` instead of `bootcmd`.

U-Boot bootcount and bootlimit (2)

- When Linux has booted successfully, a userspace application can take care of setting `upgrade_available` and `bootcount` back to `0`.
- Otherwise, the system will be rebooted either manually (pressing reset) or automatically (if a watchdog is used), and `bootcount` will increase until `bootlimit` is reached.
- Note: U-Boot's configuration offers other possibilities than storing the boot count in the environment: in an I2C device (like an RTC), in an EEPROM, in SPI flash, in a file on an EXT filesystem, in RAM (area persistent after resets)...

See `doc/README.bootcount` in U-Boot documentation for details.

Accessing the U-Boot environment from Linux

To set the `upgrade_available` and `bootcount` U-Boot variables from Linux, to start the boot counting mechanism, or to disable it when the latest update booted to completion, you can use the the `fw_printenv` and `fw_setenv` commands.

- Such tools can be built from U-Boot's source tree:
  `make CROSS_COMPILE=arm-linux- envtools`
- Yocto Project: `libubootenv` recipe
- Buildroot: `BR2_PACKAGE_UBOOT_TOOLS_FWPRINTENV` configuration
- See `tools/env/README` and
  https://elinux.org/U-boot_environment_variables_in_linux for details.

Using a watchdog

Here's the typical use of a watchdog to handle failed updates

- You configure U-Boot to enable a watchdog on your platform
- When Linux is booted, a periodic userspace process should be running to keep "feeding" the watchdog by writing to `/dev/watchdog`, otherwise the hardware watchdog will reboot the machine, and `bootcount` will be increased.
- See `watchdog/watchdog-api` in kernel documentation for details.

Editing a partition's bootable flag from U-Boot

After an unsuccessful boot, here are U-Boot commands to modify partition tables on block storage:

- `mbr` command for MBR partitions (`CONFIG_CMD_MBR`):
  - Example usage:

    ```
    => setenv mbr_parts 'name=boot,start=4M,size=128M,id=0x0e;
        name=rootfs1,size=256M,bootable,id=0x83;
        name=rootfs2,size=256M,id=0x83;
        name=data,size=-,id=0x83'
    => mbr write mmc 0
    ```

  - `mbr verify` command to check that partitions are in sync with `mbr_parts`.
  - See `https://u-boot.readthedocs.io/en/latest/usage/cmd/mbr.html` for details.
- `gpt` command for GPT partitions (`CONFIG_CMD_GPT`):
  - Similar `gpt write` and `gpt verify` commands
  - Details on `doc/README.gpt`

Why partition flags?

Why didn't we store the number of the partition to use in a U-Boot environment variable?

- Because that's what the *distro boot* functionality of U-Boot is supposed to rely on.
- Because saving the environment is far from being atomic. Multiple storage blocks need to be modified. Many things could go wrong.
- You could look for other non volatile storage on your SoC or on your board to store such information.

Limitations and advice

- We have only addressed systems booted from block devices, the most common ones today. Systems booting from NAND or NOR flash could store the partition to use on a specific non volatile storage.
- Not sure whether the provided U-Boot or even the Linux commands provide atomicity guarantees.
- At least a good idea is to use U-Boot's *redundant environment* features. If one of the two copies of the environment is corrupted, U-Boot will use the other one.

Strategies for updating the bootloader?

- Need to configure the SPL to load U-Boot from a filesystem.
- However, it's probably difficult to implement partition selection and boot counting in the SPL (space limited).
- If you're booting from eMMC, you may flash the SPL and U-Boot on the special `boot0` and `boot1` hardware partitions. See details about the `mmc partconf` command in U-Boot's documentation. To be able to update the SPL too, this requires your SoC to support this mechanism though. Otherwise, it's advisable not to update the SPL on the field.

What to remember

- U-Boot offers a `part list` command to find the block partition with the `bootable` flag.
- U-Boot offers `mbr` and `gpt` commands to edit partitions on block devices.
- U-Boot can keep track of boot failures, can count them (`bootcount` environment variable), and run an alternative boot sequence (`altbootcmd`) when a maximum number is exceeded (`bootlimit`).

Special thanks

- Vyacheslav Yurkov
  For sharing knowledge, asking questions and reviewing these slides
- Thomas Petazzoni
  For his insightful blog post about using *swupdate* on stm32mp1:
  `https://bootlin.com/blog/building-a-linux-system-for-the-stm32mp1-remote-firmware-updates/`
- Köry Maincent
  For sharing his experience

References

- This ELC 2022 presentation from Leon Anavi:
  How to Choose a Software Update Mechanism for Embedded Linux Devices
  (Video)
- Bootlin blog posts
  - swupdate: Building a Linux system for the STM32MP1: remote firmware updates
  - RAUC: Another system update adventure with RAUC, Barebox & Yocto Project
  - Mender: Mender: How to integrate an OTA updater
- Collection of *Embedded Linux Conference* talks on firmware update:
  `https://elinux.org/Upgrades_Presentations`

Questions? Suggestions? Comments?

Michael Opdenacker

michael.opdenacker@bootlin.com

https://fosstodon.org/@MichaelOpdenacker

https://bootlin.com/pub/conferences/2022/elce/