Journey of EBBR compliance on NXP Layerscape devices

Priyanka Jain <priyanka.jain@nxp.com>
Poonam Aggrwal <poonam.aggrwal@nxp.com>
Ilias Apalodimas <ilias.apalodimas@linaro.org>

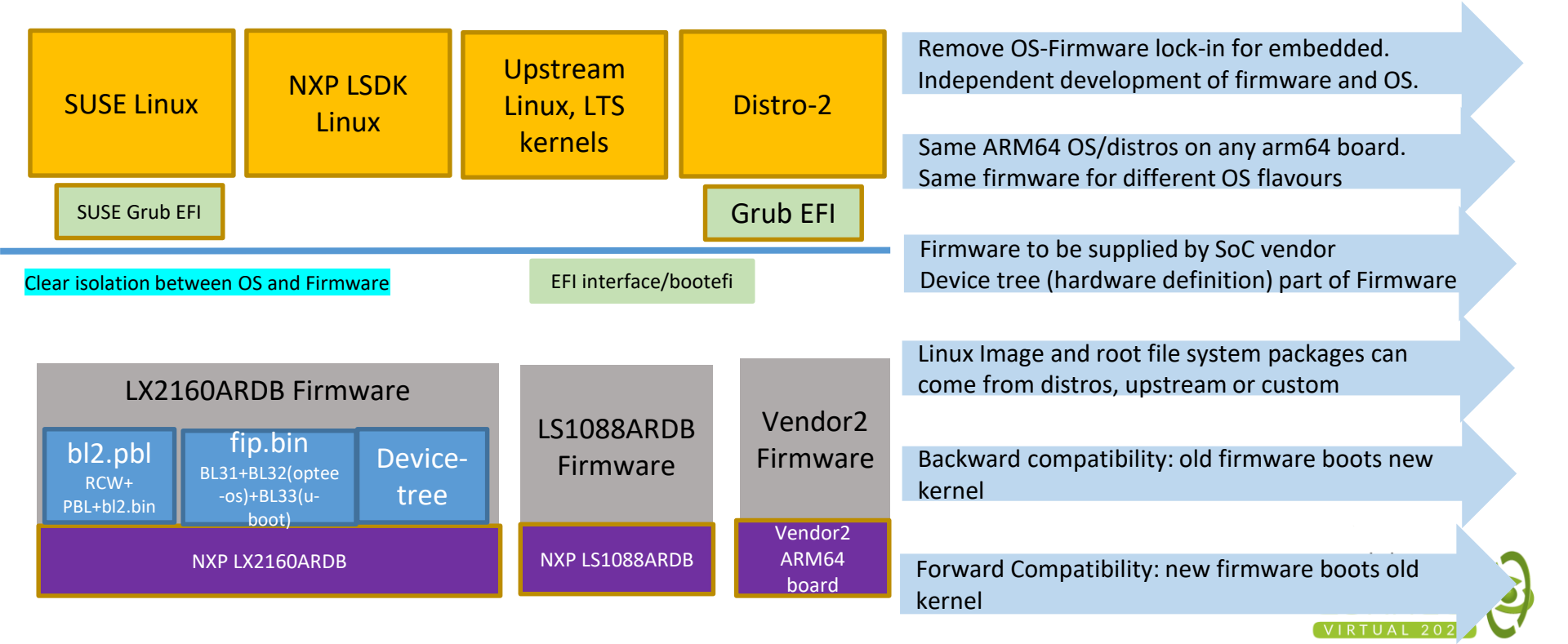
Linaro Connect VIRTUAL 2020

AGENDA

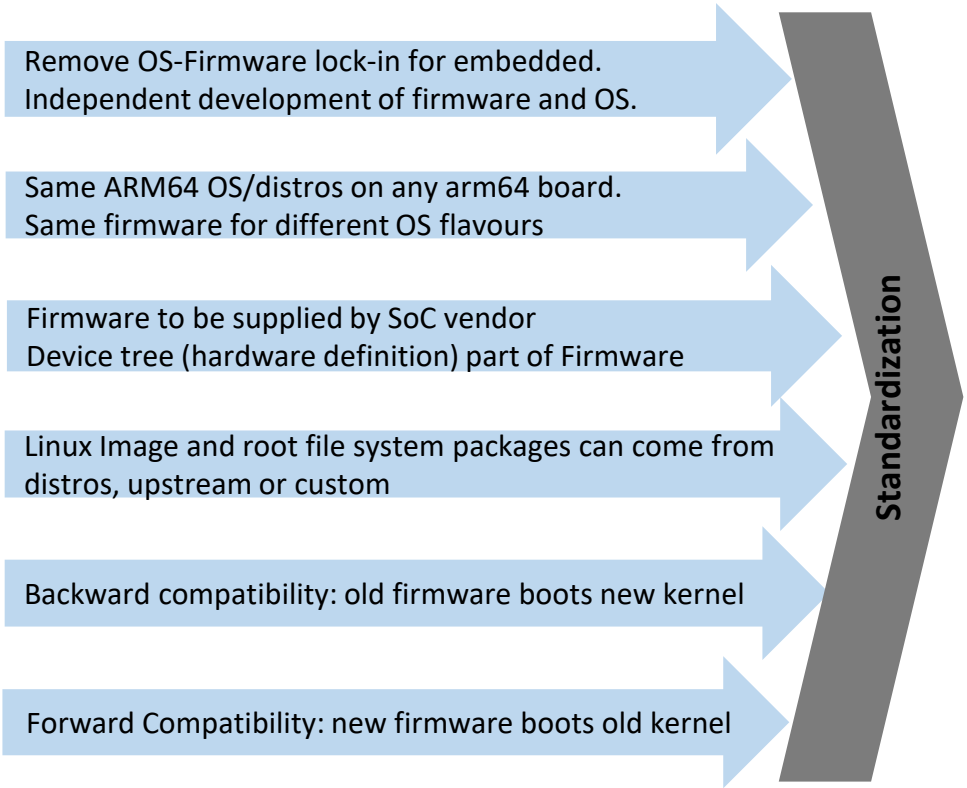
1. Why EBBR
2. EBBR status on NXP Layerscape devices
3. Challenges enabling EBBR on NXP Layerscape devices
4. Secure Variable support using RPMB & its status on NXP Layerscape device
5. Examples of EBBR running on NXP Layerscape device (Demo)

Why EBBR?

Embedded Base Boot Requirements



Why EBBR?



EBBR(Embedded Base Boot Requirement) Specification

Specification to define requirements on platforms and OS for Embedded Products

EBBR status on NXP Layerscape devices

- NXP Layerscape software follows upstream first.
  - Platforms are supported in upstream. Hence, all upstream development (efi) can be easily used on NXP products
- Various OS like upstream Linux, SUSE distro(announce Layerscape platforms in releases), LSDK Linux can boot on same NXP firmware.
- OS & Firmware Images:
  - NXP releases its firmware as part of LSDK Releases as well as upstream it. Device tree used is from Linux repository.
  - Linux Image and root file system packages can come from distros, upstream or custom.
- Backward Compatibility(old firmware boots new kernel): Yes. DT-bindings are backward compatible
- Forward Compatibility (new firmware boots old kernel): currently a challenge

EBBR Sepcification (v1.0.1)	NXP Firmware (u-boot based)
Block Device Partitioning (firmware must support MBR, GPT and El Torito partitioning on block devices)	support distro boot (USB/SATA/SD) ✓
Firmware storage Firmware and OS images on different partitions/devices	GPT partitions for firmware(SATA/USB, SD for non-SD boot), MBR (SD for SD-boot) Fixed Shared Storage: eMMC, Flexspi-boot via on-board flashes Removable shared storage: Sd-boot Filesystem firmware images: not supported[Recommended] ✓
UEFI executes as 64-bit code at either EL1 or EL2	U-boot(load EFI) runs at EL2 ✓
RAM defined by the UEFI memory map must be identity-mapped	U-boot identity-mapped RAM ✓
Configuration Table(ACPI/DT)	U-boot is Device Tree based ✓
UEFI boot services *(EFI_GET_VARIABLE,GET_NEXT_VARIABLE, SET_VARIABLE, RESET_SYSTEM)*	U-boot supports bootefi ✓
UEFI runtime services *(EFI_SET_VIRTUAL_ADDRESS_MAP, EFI_CONVERT_POINTER)*	OS booted via bootefi and virtual mapping is supported ✓
UEFI runtime services *(EFI_GET_VARIABLE, GET_NEXT_VARIABLE) (optional)*	upstream u-boot ✓
Runtime Device mapping (Firmware runtime mapping should not interfere with OS)	Yes ✓
EFI Reset and Shutdown	TFA (PSCI) ✓
Secure Firmware (System Boot, SMP-boot, Power)	TFA (PSCI) (no spintable) ✓

Challenges enabling EBBR on NXP devices

Filesystem based firmware This requires change in SoC Boot Architecture flow
Not supported , since the NXP Boot ROM cannot interpret file system.

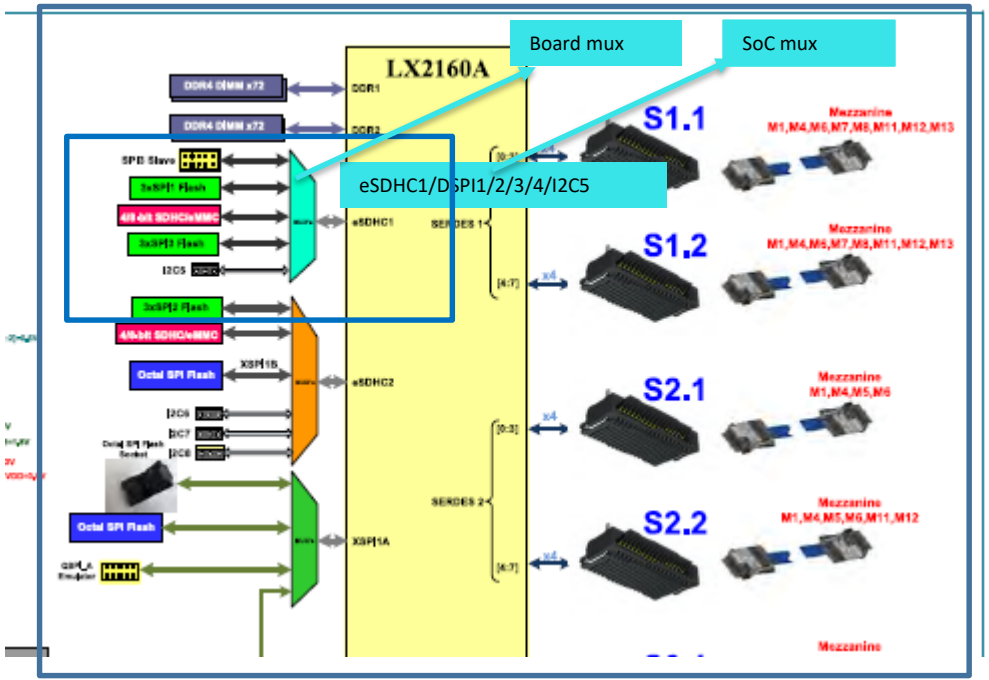
For SD boot, GPT can't (only MBR can) be supported for SD device. The boot ROM expects the first data structure at 0x1000, which is not enough to support GPT partitions.

Device Tree should be part of firmware and not Linux

- NXP U-boot and Linux have separate DTs. It should be just one which captures the hardware description. Distros (like SUSE)prefers u-boot device tree to boot OS.
- This is a community wide accepted problem which needs to be resolved.
- This is already actively worked by EBBR owner (Grant) and Linaro in Device tree forums.
- Some proposals on the table: separate repo for Device tree, only u-boot device tree, System Device tree.
- Upstream u-boot DT patches are accepted from the stable commit of Linux to match u-boot and Linux DTs
- Backward compat for Linux DTs is strictly followed (newer kernels on old DTs)
- Any compatibility breaks should be informed to the community

Run time device tree fix-ups for NXP SoCs

- NXP SoCs provide signal multiplexing to provide various interfaces using RCW configuration.
- Single device tree is used for different mux configuration.
- Based on RCW read /some hwconfig /or run time detection mechanisms uboot fixes up the DT
- LX2 example shown below.



File: board/freescale/lx2160a/lx2160a.c

```
int esdhc_status_fixup(void *blob, const char *compat)
{
/* Enable esdhc and dspi DT nodes based on RCW fields */esdhc_dspi_status_fixup(blob);
return 0;
}
```

esdhc_dspi_status_fixup

```
/* Check RCW field sdhc1_base_pmux to enable/disable
 * esdhc0/dspi0 DT node
 */
sdhc1_base_pmux = gur_in32(&gur->rcwsr[FSL_CHASSIS3_RCWSR12_REGSR - 1])
                                        & FSL_CHASSIS3_SDHC1_BASE_PMUX_MASK;
sdhc1_base_pmux >>= FSL_CHASSIS3_SDHC1_BASE_PMUX_SHIFT;
if (sdhc1_base_pmux == SDHC1_BASE_PMUX_DSPI) {
                        do_fixup_by_path(blob, dspi0_path, "status", "okay", sizeof("okay"), 1);
                        do_fixup_by_path(blob, esdhc0_path, "status", "disabled",sizeof("disabled"),
1);
} else {
                        do_fixup_by_path(blob, esdhc0_path, "status", "okay", sizeof("okay"), 1);
                        do_fixup_by_path(blob, dspi0_path, "status", "disabled",
                                                                        sizeof("disabled"), 1);
```

config_board_mux function configures the board muxes

Secure Variable support using RPMB

- Embedded Devices don’t have secure world flash, but they do have RPMB
  which can be used to store efi variables
- EDK2: StandAloneMM to verify/read/store EFI variables

- OPTEE-OS: can read/store data in RPMB partition
  - Code is added to launch StMM from SPM(Secure Partition Manager) within OPTEE
- Benefits :
  - Re-use StMM
  - Match > Arm 8.4 FFA architecture
  - Store EFI variables in RPMB
- Limitation :
  - SPD (Secure Payload Dispatcher) is used for OP-TEE and unfortunately SPD and SPM are mutually exclusive.
  - So one can’t have EFI variables and a TEE OS at the same time.

Get/SET EFI variable status in Linux

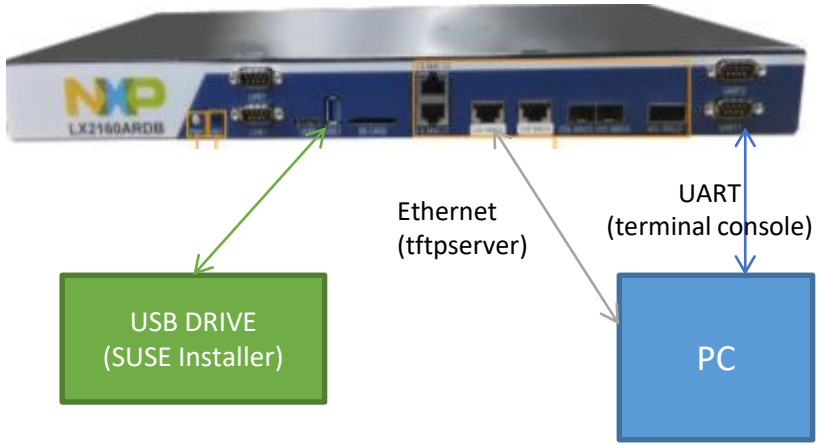
- GetVariable and GetNextVariable are now supported in Linux
- SetVariable still returns EFI_UNSUPPORTED
  - GetVariable operates on a memory backend
  - Due to this, fixing SetVariable is a bit more complicated

Examples of EBBR running on NXP devices

1. Boot Linux(NXP LSDK release, upstream) via bootefi command on LX2160ARDB
2. Boot Linux via grub.conf on LX2160ARDB
3. Boot Distros like Suse on LX2160ARDB,
4. Traverse, an Australian OEM : Ten64 appliance(with LS1088A) can boot Suse
5. GetVariable, GetNextVariable, SetVariable supported in u-boot. Tested on LX2160ARDB
6. GetVariable, GetNextVariable supported in Linux. Tested on LX2160ARDB

Demo1: SUSE DISTRO Booting on LX2160ARDB

1. Copy openSUSE-Installer on storage device(eg USB drive) connected on board
   a) *dd if=openSUSE-Tumbleweed-DVD-aarch64-Current.iso of=/dev/sdb*
2. Flash and boot with upstream v2020.07u-boot
3. Do device tree fix-up. DTB: LSDK 20.04/upstream Linux
   - *tftp 0x90000000 fsl-lx2160a-rdb.dtb*
   - *fdt addr 0x90000000; fdt boardsetup;*
4. Deploy mc
   - *fsl_mc apply dpl 0x20d00000*
5. Execute Boot command in u-boot -> distro boot starts -> Suse Installer



Demo2: Get/Set secure variables (via optee)

1. Get sources :
   - Upstream u-boot, Linux; optee-os + optee_osdiff
   - LSDK 20.04 TFA, RCW (https://lsdk.github.io/)
   - edk2, edk2-platforms + edk2-platformdiff

2. Refer NXP LSDK 20.04 Release user guide for building images

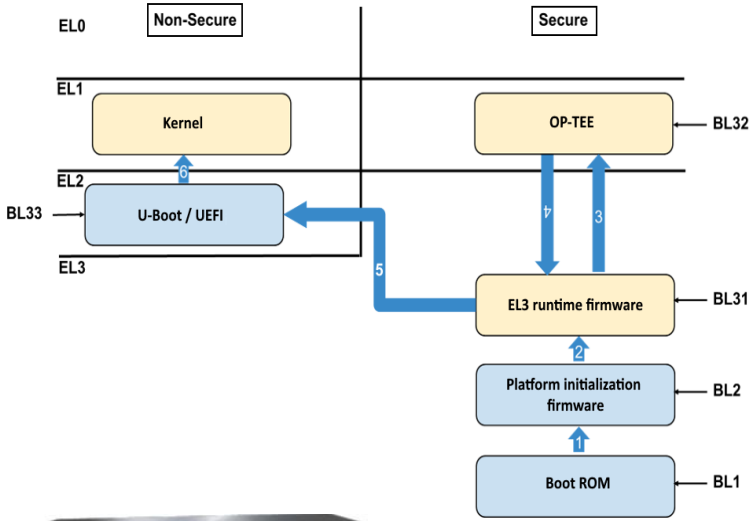
   Optee-os needs additional step to build with STMM image

   #make CFG_ARM64_core=y PLATFORM=ls-lx2160ardb CFG_STMM_PATH=BL32_AP_MM.fd CFG_RPMB_FS=y CFG_RPMB_FS_DEV_ID=1 CFG_CORE_HEAP_SIZE=524288 CFG_RPMB_WRITE_KEY=1

3. On board
   1. Boot board with built fip(BL31, BL32: optee, BL33:u-boot), bl2.pbl(RCW + PBL cmd +bl2.bin) image
   2. Get/Set secure efi variables in U-boot

      #printenv –e

      #setenv –e bs –rt –nv test test1
   3. Boot kernel via bootefi after device-tree fixups;get secure efi variables *#efivar -l*



Thank you

Accelerating deployment in the Arm Ecosystem

Linaro Connect VIRTUAL 2020