S32 Linux® BSP Product Brief

1.0 Software Product Overview

NXP Automotive Linux® BSP follows the general layout of a BSP, containing a bootloader (U-Boot) the Linux kernel, a root file system, which can contain various libraries and middleware, and sample applications. The objective is to enable more hardware platforms, to build on top of the AP Linux BSP and add supplemental components such as drivers or applications.

More details are provided in Chapter 2.0.

For all open source components included in the BSP, if community versions of the components exist, they are reused in the Linux BSP, after applying the necessary modifications to enable execution on NXP hardware. The modifications might include: specific configuration of components, driver modifications, device tree definitions for NXP specific hardware, new driver and feature implementations, writing new sample applications, etc.

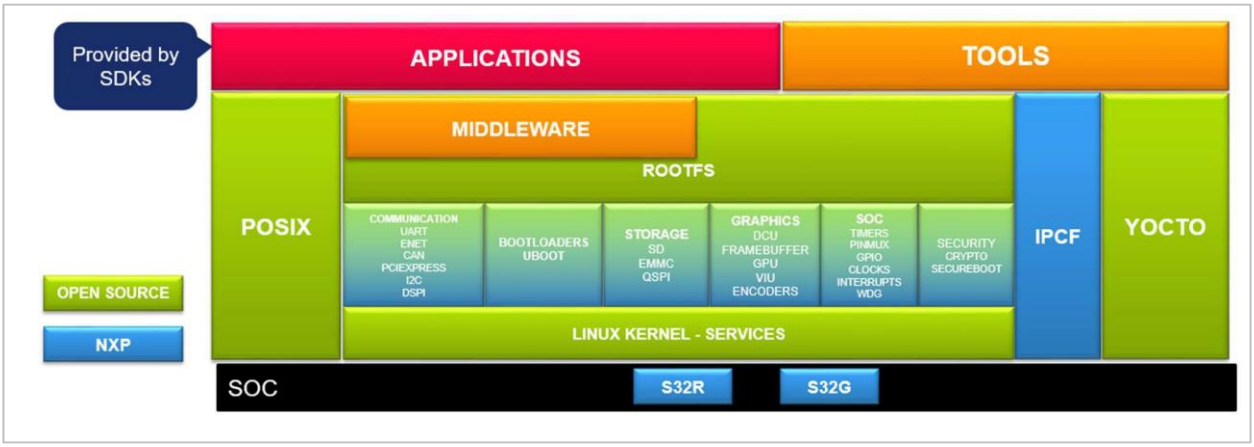

Figure 1 Linux BSP Architecture Diagram

2.0 Software Content

This section contains a description of the Linux BSP features covered by the following main Linux BSP architecture components:

- ARM® Trusted Firmware
- U-Boot
- Linux Kernel
- Yocto
- ROOTFS
- Drivers
- IPCF
- POSIX

2.1 ARM Trusted Firmware

The ARM Trusted Firmware (TF-A) is a software component of the Linux BSP with a double role as a boot loader and Secure Monitor. As a boot loader it runs before U-Boot on the ARMv8 boot core and performs core and SoC initializations. As a Secure Monitor the TF-A provides power management services to other software components, via the standard Power State Coordination Interface (PSCI) API/ABI.
The TF-A is currently only offered on a subset of the platforms supported by the Linux BSP.

2.2 U-Boot

U-Boot is a boot loader responsible for the initialization of specific hardware components to a basic level, enough to allow the loading of the Linux kernel image, the device tree blob and, if used, an initial RAM file system (initramfs or initrd) from a boot media (such as an SD card, eMMC, QSPI flash, network, etc.)
Once the necessary images are loaded into memory, U-Boot will hand over control to the Linux kernel. After the kernel is started, U-Boot has finalized its job and the memory it has used will be reused by the kernel according to the memory management policies Linux has in place.

2.3 Linux Kernel

The Linux kernel is responsible for memory and resource management, scheduling, driver loading and unloading, interrupt handling and providing APIs (syscalls, ioctls, pseudo or virtual filesystems) for communication between the kernel or kernel drivers and user space applications.
The Linux kernel from the Automotive Linux BSP (ALB) is a version mostly comprised of a specific community version (the base version) and ALB-specific additions to allow the kernel to enable NXP hardware to work.
The Linux kernel source also provides a reference implementation of the drivers, showcasing the way the driver for a specific peripheral should work, in case an implementation on another operating system is desired.

2.4 Yocto

The Yocto component is responsible for providing a way of building all BSP components and tools necessary to create the binary images that run on the target NXP hardware.
It gives developers and clients the flexibility to supplement the provided images with other tools, binaries and utilities that should be part of the final ROOTFS (root file system) on the target hardware.

2.5 ROOTFS

The ROOTFS (root file system) represents all of the BSP's internal files together with the hierarchy of directories used to organize them. It contains both system files (kernel image, binary drivers, system configuration files, etc.) as well as user files (custom applications, etc.) and tools.

2.6 Drivers

Kernel drivers offer a hardware abstraction layer by implementing Linux kernel APIs and provide mechanisms for user space applications to interact with the hardware (usually indirectly, through libraries). The interaction paths are driver and hardware dependent, so they can differ from driver to driver. Exact specifics on the communication, capabilities and limitations shall be provided in other documents such as Release Notes, user manuals or documentation included in the source code.
These drivers are part of the same source as the Linux kernel, but, in binary form (.ko modules) can be provided separately from the kernel image.

2.7 IPCF

The Inter-Platform Communication Framework (IPCF) is a set of drivers and libraries bundled together to provide a common communication channel between separate processing units (individual cores belonging to the same or separate SoCs), over a variety of hardware communication media (such as Ethernet, PCI Express or shared memory).

2.8 POSIX

The Portable Operating System Interface (POSIX) represents a set of standards aimed at providing compatibility between various operating systems. They are comprised of a series of APIs, together with various utilities available inside the root file system. Being a Linux system, the Linux BSP is mostly POSIX-compliant.

3.0 Supported Targets

The software described in this document is intended to be used with NXP Semiconductors S32G2 devices.

4.0 Quality, Standards Compliance and Testing Approach

Linux BSP product is developed according to NXP Software Development Processes that are Automotive-SPICE (tailored for open source), IATF16949 and ISO9001 compliant.