SAMSUNG ARTIK™ Modules

Linux Architecture Guide

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

VERSION HISTORY

Revision	Date	Description
V1.0	October 19, 2016	ARTIK Linux Architecture Guide
V1.01	January 30, 2017	Updated Layout. Included Platform Support section.
V1.02	February 07, 2017	Updated Layout.
V1.03	August 31, 2017	Added Ubuntu details. Updated Linux kernel 4.4 information.

Platform Support

This document describes the basic architecture of the Linux® operating system that is present on a variety of Samsung ARTIK™ Modules. The following Samsung ARTIK™ Modules do have a Linux® operating system:

Platform	Version
ARTIK 520 Module	No Limitation, holds for all released versions
ARTIK 530 Module	No Limitation, holds for all released versions
ARTIK 710 Module	No Limitation, holds for all released versions
ARTIK 1020 Module	No Limitation, holds for all released versions

INTRODUCTION

The Samsung ARTIK™ Linux® Architecture Guide will describe the components that comprise the software architecture that is in place on various ARTIK Hardware Modules.

The first section will start with a description of the Linux® kernel and its components. The Linux® kernel contains the Kernel Core, Drivers and BSP packages.

The next section will describe the boot loader U-Boot. The bootloader is responsible for loading the default operating system that is available on the Hardware Modules during shipping.

After the bootloader the filesystem RootFS will be discussed. The file system contains System, Multi Media, Graphics, Connectivity, Network Protocols, Libraries and Development Modules.

This document should give the informed Linux® developer insight into the inner workings of the Linux® software stack residing on the referenced Hardware Modules.

Overview

The Linux-based OS running on ARTIK 5/7 family of modules is a customized version of Fedora/Ubuntu distributions.

The Fedora Linux® operating system is based on the latest version of the open source code of Fedora project. Ubuntu is also a Linux OS based on the Debian architecture. It is an open source software OS that is available for desktop, servers, and IoT devices.

Our Hardware Modules however use a custom light weight version of Fedora which is built from scratch and only has the essential packages needed for IoT deployment. Our specialized implementation contains 3 major components:

- Linux® kernel
- U-Boot (Boot Loader)
- RootFS (Root File System)

Figure 1. Basic Hardware Module Architecture shows the components pictorially. The subsequent sections will describe each component in greater detail.

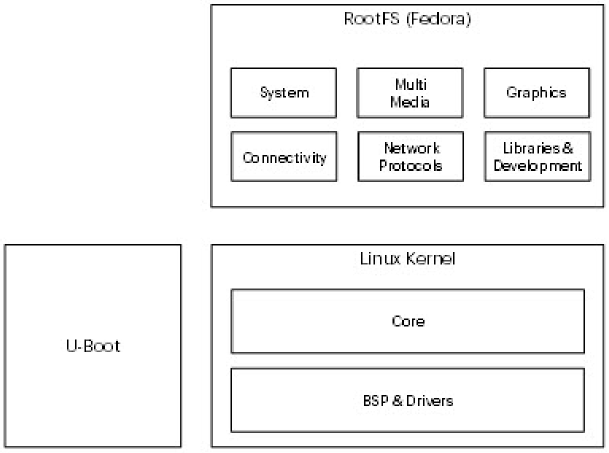

Figure 1. Figure 1. Basic Hardware Module Architecture

LINUX® KERNEL

The current Linux kernel version present on our Hardware Modules is based on the Linux version 4.4. We consolidated the Linux® kernel version and its associated source tree to assure support for our Hardware Modules from one single source. The hardware used is fully described in the device tree specification file that will be located in the "/boot" partition of the Hardware Modules.

The basic version rule of the ARTIK kernel is the "LTS" (Long Term Support) kernel. The LTS kernel can support critical bug fixes such as security patches till it reaches End Of Life. The 4.4 kernel will be maintained till Feb, 2018. Linux® Kernel Architecture shows the various Linux® versions with its associated EOL timing. See *Table 1*.

Table 1. Linux® Kernel EOL Structure

Version	Maintainer	Release	Project End Of Life
4.4	Greg Kroah-Hartman	2016-01-10	Feb, 2018
4.1	Sasha Levin	2015-06-21	Sep, 2017
3.18	Sasha Levin	2014-12-07	Jan, 2017
3.16	Ben Hutchings	2014-08-03	Apr, 2020
3.14	Greg Kroah-Hartman	2014-03-30	Aug, 2016
3.12	Jiri Slaby	2013-11-03	Jan, 2017
3.10	Willy Tarreau	2013-06-30	Oct, 2017

Linux® Kernel Architecture

Figure 2 shows a picture of the Linux® architecture with its main components. The Linux® kernel consists of the kernel core the drivers and the BSP. The kernel core provides the network stack, power management, file system, memory management, scheduler and the block layer. The driver & BSP initialize H/W resources from device drivers and the board support package. They also contain USB, sound, graphics, 802.11 and the Bluetooth® subsystem. shows the Linux architecture.

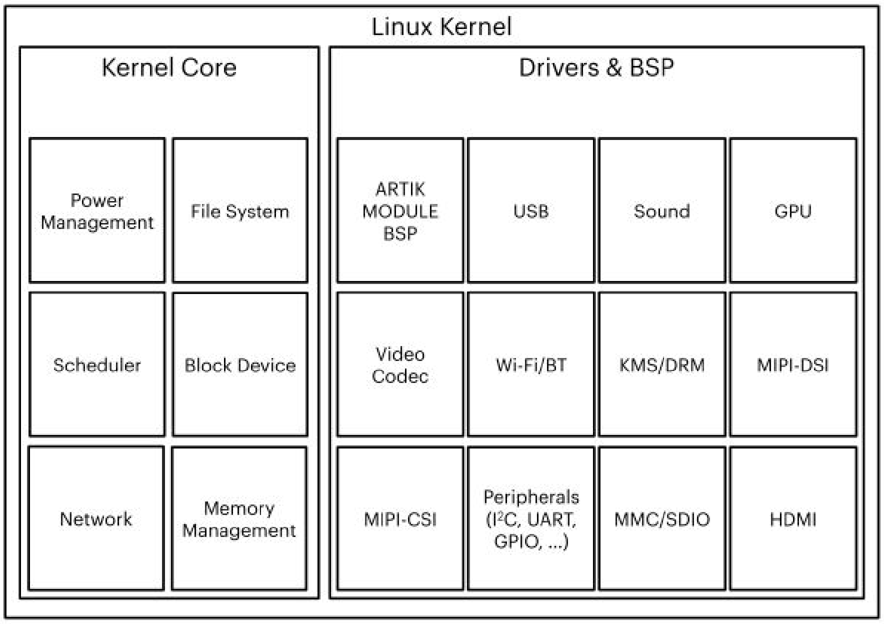

Figure 2. Linux® Kernel Architecture

The kernel binary consists of three major components namely:

- zImage
- device tree blobs
- kernel modules

zImage

The zImage is the compressed kernel binary that will be decompressed during the early stages of the booting process. The zImage includes prebuilt device drivers.

Device Tree

The device tree blob is the data structure that describes the hardware.

Kernel Modules

The kernel modules are located in the "/lib/modules" directory. They can be loaded/unloaded into the kernel upon demand. These kernel modules can also reduce prebuilt kernel binary size and kernel booting time.

The ramdisk can be used when an eMMC recovery from micro sdcard is required see *Figure 3*. A special "sdfuse" image contains the image that can be recovered from ramdisk using U-Boot. During normal booting, the ramdisk is not used and the kernel will mount the Fedora root filesystem directly.

The "boot/rootfs" binaries are stored in eMMC. *Figure 3* describes the eMMC disk layout of the supported Hardware Modules. The Hardware Modules do not use the eMMC boot partition, like the generation-1 ARTIK Modules did, because the boot rom code cannot load the "bl1" binary from the eMMC boot partition. To circumvent this drawback, the bootloader ("bl1","bl2" and U-Boot) is stored in the normal eMMC partition.

The factory information (such as the serial number) is also stored in the eMMC partition. This information can be accessed using U-Boot. There are three partitions that can be accessed from the Fedora root file system:

- The first partition is the FAT partition that stores the kernel binary, device tree blob and the ramdisk image.
- The next partition is the mounted "/lib/modules" directory storing kernel loadable modules. These modules will be loaded on demand from the Fedora root filesystem.
- The last partition is “RootFS”. This partition occupies residual storage of the eMMC.

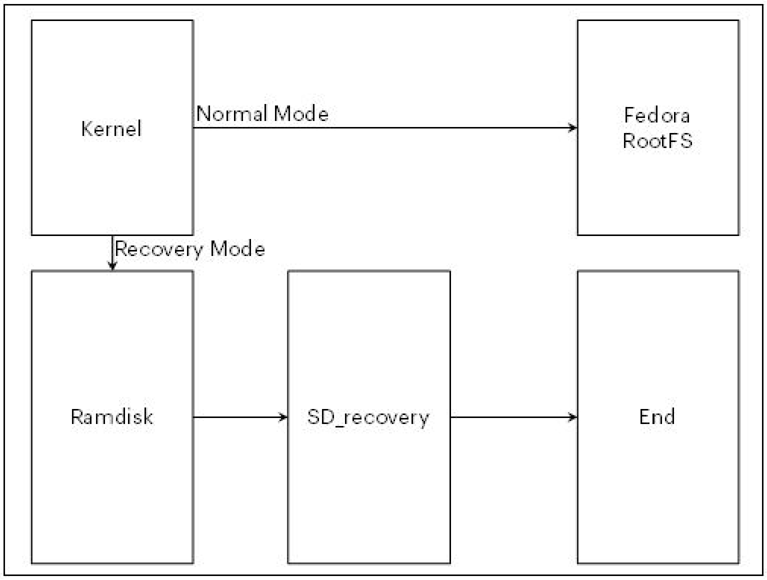

Figure 3. Recovery Boot Sequence

BOOTING PROCESS

Boot Sequences

Figure 4 shows the partition tables of both a normal boot using the eMMC Partition Table and a boot from the SD Card using the SD Card Partition Table.

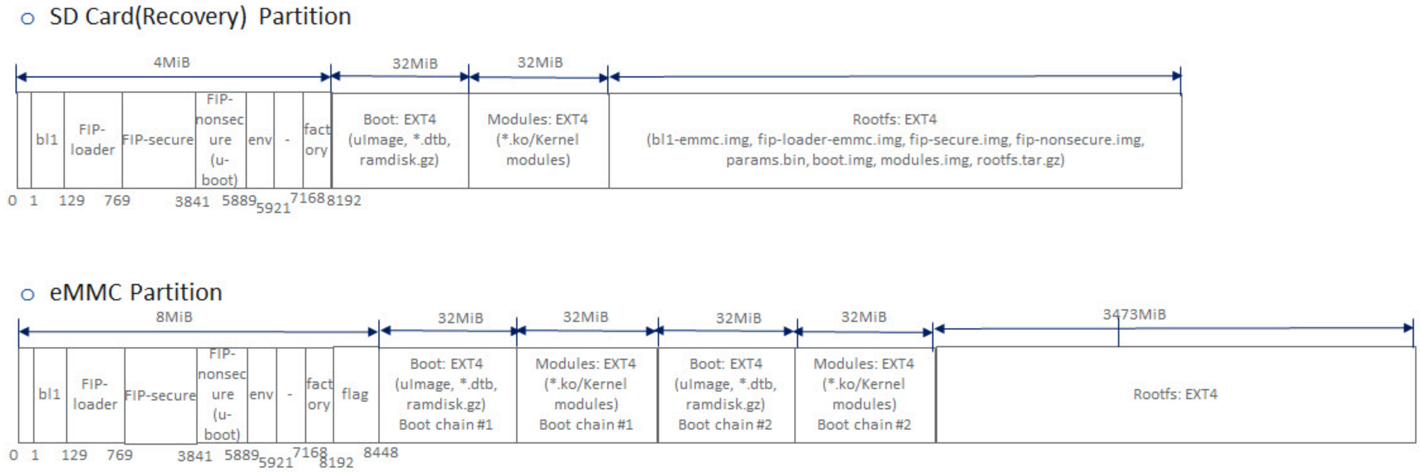

Figure 4. eMMC Normal Partition Table

The Partition Tables are split up into a Bootloader Partition and a User Partition. The components that can be found in subsequent partitions are:

1. Bootloader Partition
   a) First Sector : DOS Partition Table.
   b) bl1 : The first Bootloader of the Hardware Modules. The content is loaded into SRAM.
   c) FIP-loader : This loader loads the FIP Binary.
   d) FIP-secure : Secure bootloader that will initialize the secure area.
   e) FIP-non secure: U-Boot partition.
2. User Partition
   a) "/boot" (Read only): The kernel images such as uImage and tree blobs.
   b) "/lib/modules" (Read only) : This partition contains the kernel modules. In addition the root file system will mount "/lib/modules".
   c) "/" (Read/Write) : Root file system.

eMMC Boot Sequence

When the Hardware Module is powered on, the boot rom code "bl0" inside the Hardware Module chip will load the "bl1" binary from eMMC into on-chip SRAM. This "bl1" binary is located in the first sector offset, once loaded it will start executing "bl1" code. The "bl1" code, amongst other responsibilities, loads the FIP-loader into DRAM and once loaded jumps to the FIP-Loader code.

The FIP-loader binary loads the ARM® Trusted Firmware (ATF), located in the FIP-secure partition, and U-Boot, located in the FIP-non secure partition, into the proper DRAM locations. The FIP-secure binary contains the ATF BL31 used for handling secure monitor calls and the ATF BL32 for secure OS calls (OPTEE). Once these images are loaded the FIP-loader will jump to the U-Boot entry point.

The U-Boot loader that loads the kernel binary and the device tree blob from the "/boot" partition, will jump to the Linux® kernel and the kernel core/device drivers will be initialized during kernel booting. Once device drivers are initialized the RootFS system starts. This system is the first process that will maintain boot daemons such as "pulseaudio" and a Bluetooth® daemon. *Figure 5* shows the various steps in a typical (eMMC) booting scenario.

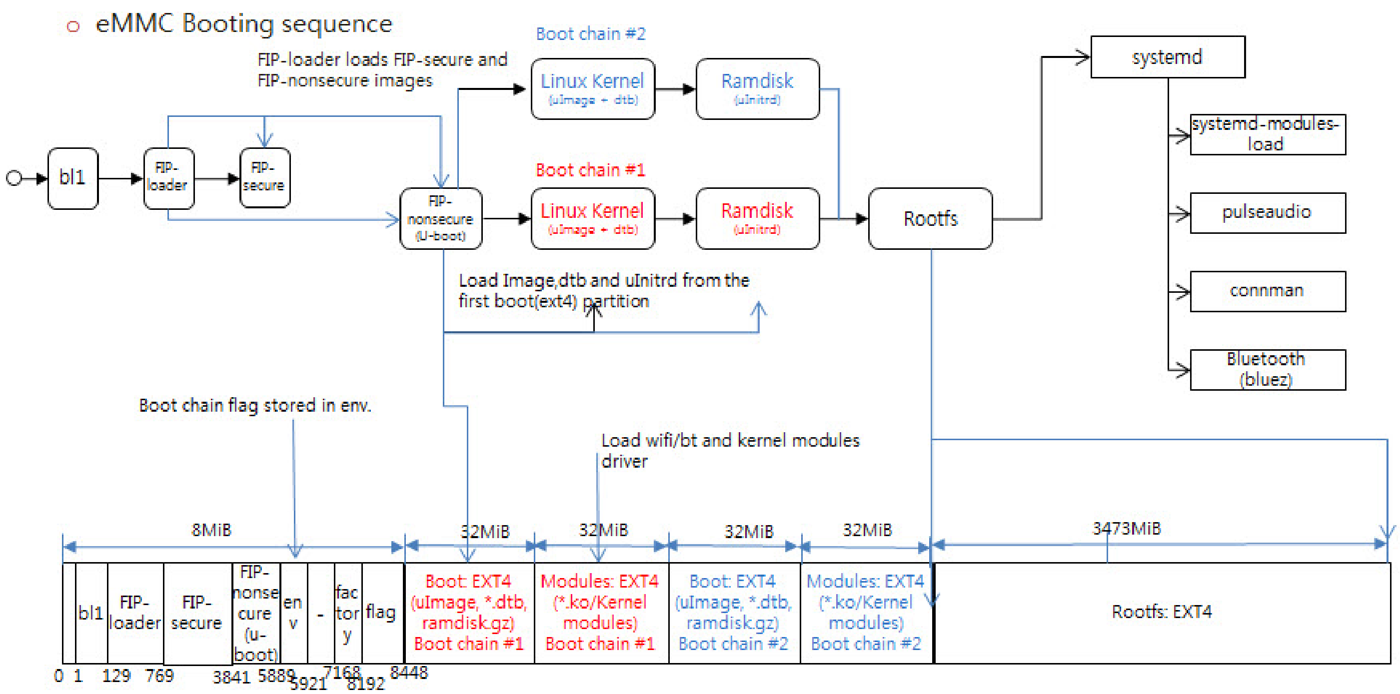

Figure 5. eMMC Booting Sequence

U-BOOT

The U-Boot is the boot loader that is initiated after the FIP-loader sequence is completed. Its primary purpose is to load the operating system (in this case Fedora Linux®). In addition to loading the Fedora OS, the U-Boot loader takes responsibility for:

- System Initialization
- Determine the Boot Mode
- Firmware Fusing or kernel binary loading
- Boot argument control : Control and pass boot parameters from and to the kernel

The U-Boot loader has environment variables that control the booting procedure. It also provides a U-Boot shell and several helper commands. *Figure 6* shows the most important tasks U-Boot is responsible for.

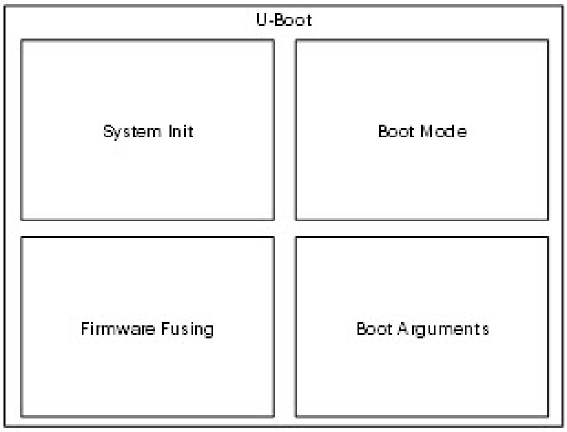

Figure 6. U-Boot Architecture

Figure 7 shows the sequence U-Boot will adhere to during the booting process. As an example the first task for U-Boot is to try to run the "bootcmd" using the U-Boot environment variables. After completion of the "bootcmd" it will follow the flow chart till the U-Boot will jump to the kernel address.

The "factory_info" ("run factoryload") will provide information such as the Ethernet mac address. Once the factory information is collected the boot arguments will be configured and delivered to the Linux® kernel.

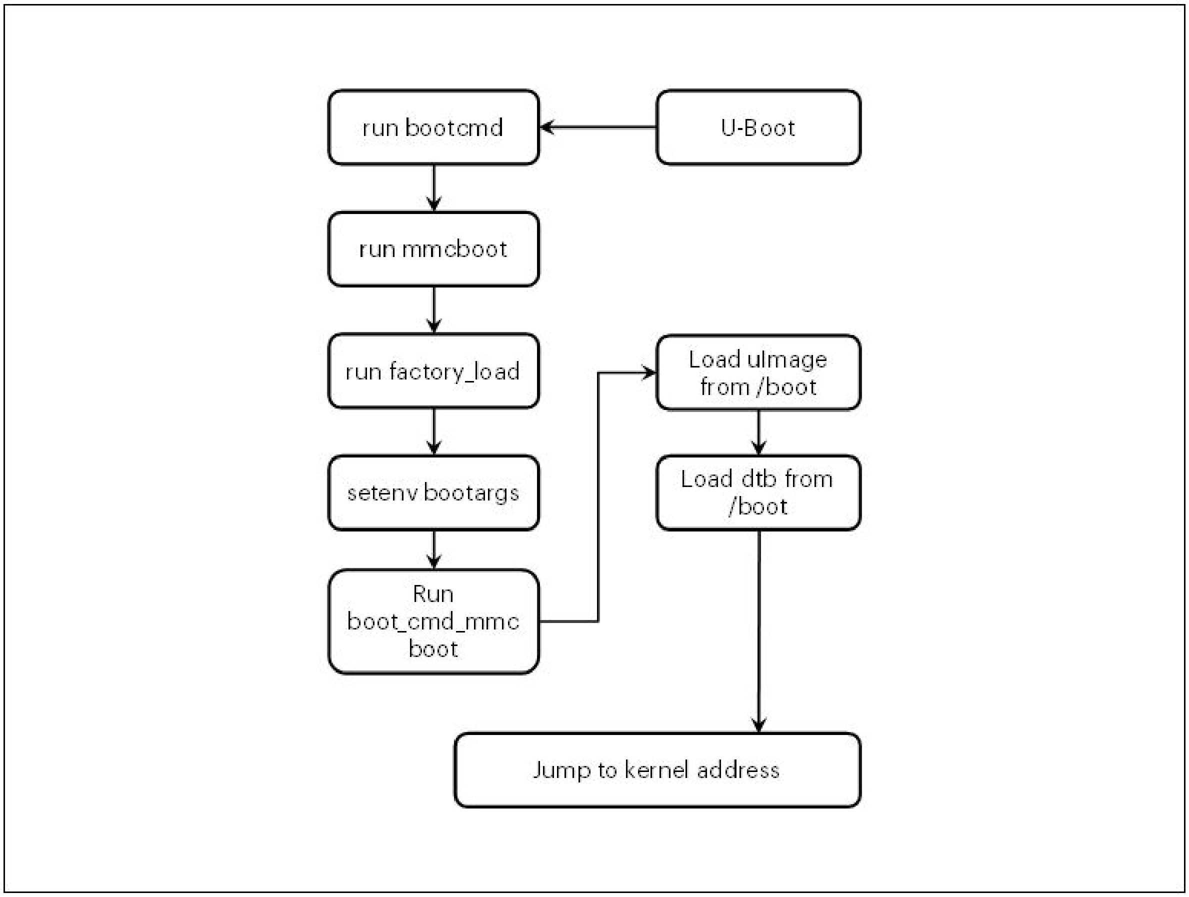

Figure 7. U-Boot Sequence

ROOTFS

The Fedora Linux® operating system used in our Hardware Modules is based on a custom light weight version of Fedora, that we custom build to assure optimal performance in a IoT environment. The RootFS system is an important component of the OS and it consist of the following components:

- System
- Multi Media
- Graphics
- Connectivity
- Network Protocol
- Libraries and Development

Figure 8 describes the main component in the RootFS system. The subsequent sections will go more in-depth on the RootFS components mentioned above.

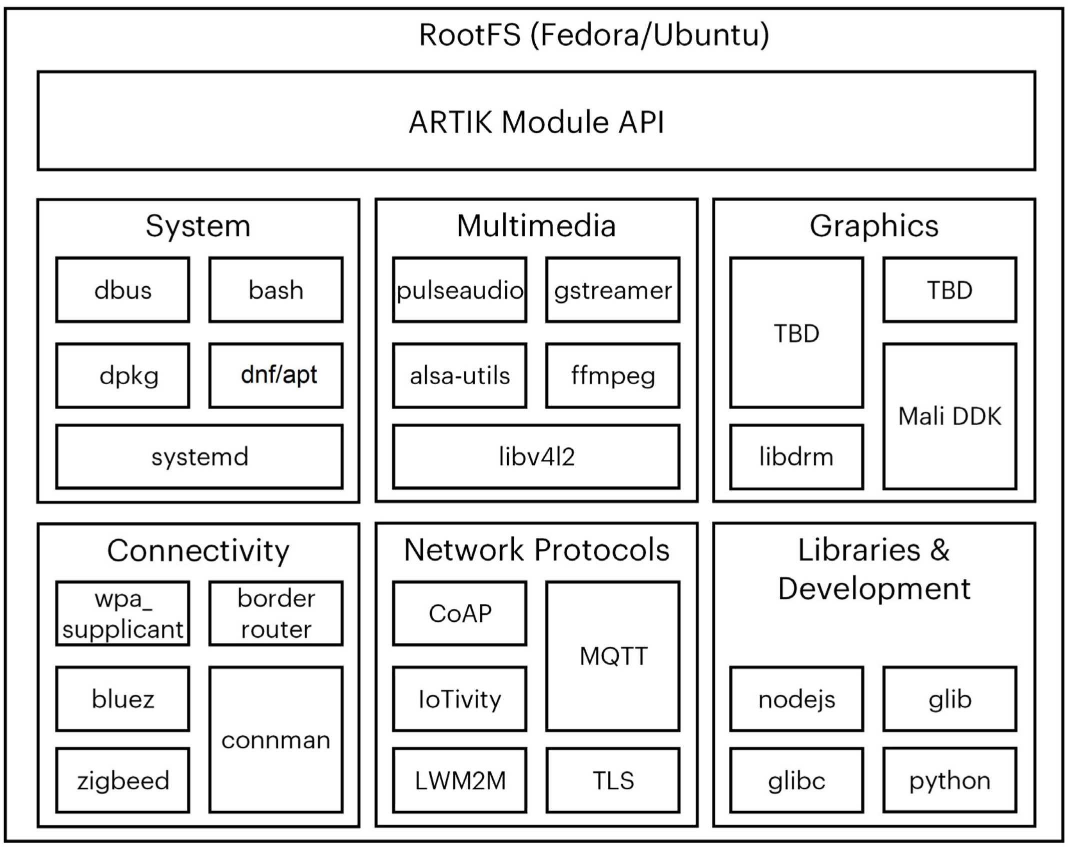

Figure 8. RootFS Architecture

System

The system group is an essential part of any RootFS (Fedora/Ubuntu). The "systemd" is an "init" daemon that can run daemons and services concurrently using initialization order. Each system service can communicate with other services using the "dbus" IPC mechanism.

MultiMedia

The multimedia system of the Hardware Modules consists of both audio and video. *Figure 9* shows the Multimedia architecture implemented on the Hardware Modules.

Audio

"ALSA lib" is a framework that provides a software API for audio device drivers. The Pulse audio module is based on the "ALSA lib" for supporting sound. Pulse audio runs a sound server. This is a background process that accepts sound from one or more sources (processes or capture devices) and redirects this sound into one or more sinks like a sound card a remote network Pulse audio server or other processes. The system configures ALSA such that it uses a virtual device provided by Pulse audio. Therefore applications using the "ALSA lib" framework will output sound to Pulse audio, which then uses the "ALSA lib" framework itself to access the real sound card.

Video

The video system in the Hardware Modules is based on the "gstreamer" and "ffmpeg" libraries and utilities. The "gstreamer" framework is a pipeline-based multimedia framework that links together a wide variety of media processing systems to complete complex workflows. A user can run a preview, a recording, or he/she can play a video stream using various "gstreamer" plugins. In addition to "gstreamer", "ffmpeg" can be used for similar type of functions. The "gstreamer" framework does support hardware accelerated video encoding and decoding as a plugin.

All "gstreamer" audio and video communicate using the "v4l2" kernel library. The "ffmpeg" library does not support hardware acceleration.

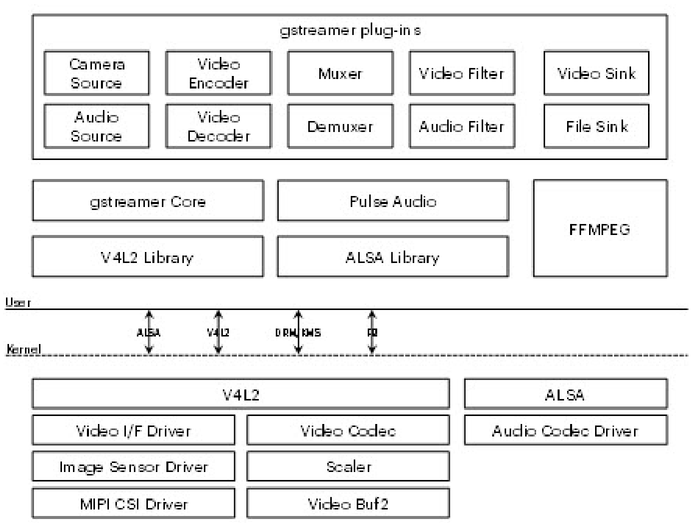

Figure 9. MultiMedia Architecture

Graphics

The graphics system of the Hardware Modules use a DRM (Direct Rendering Manager) based system called "libdrm". In addition to the "libdrm" API's that are provided as the default approach, the graphics system also provides infrastructure for a "framebuffer" approach. The Hardware Modules have ARM® Mali TM400-MP4 GPU hardware supporting Mali TM DDK with OpenGL®-ES and X11 EGL acceleration layer.

Libraries and Development

The "glibc" (GNU-C library) is the base library which can provide C-API functionality and system calls to the kernel. Many of the libraries such as Multimedia ("gstreamer") and Graphics use the "glib2" library that provides advanced data structures for programming. In addition to GNU-C library support, we also support "Nodejs" and "python" to create scripts and programs.

Connectivity

The Hardware Modules support 802.11, Bluetooth®, ZigBee® and Ethernet connectivity. *Figure 10* shows the Connectivity architecture that is currently present on the Hardware Modules.

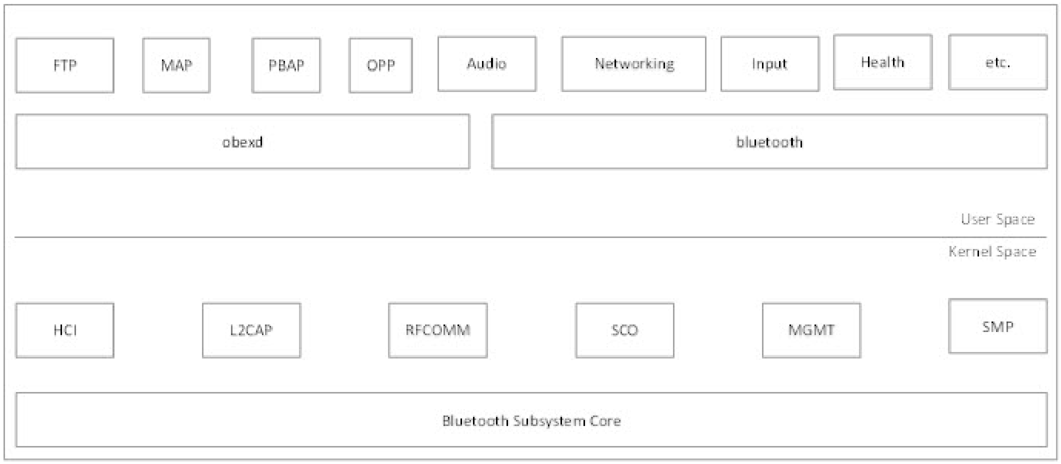

Figure 10. Connectivity Architecture

Bluetooth®

The Bluetooth® library is supported using the "BlueZ" module. The following components are part of the "BlueZ" module:

- Bluetooth® kernel subsystem core
- L2CAP and SCO audio kernel layers
- RFCOMM, BNEP, CMTP and HIDP kernel implementations
- HCI UART, USB, PCMCIA and virtual device drivers
- General Bluetooth® and SDP libraries and daemons
- Configuration and testing utilities.

ZigBee®

The ZigBee® software consists of the following major components:

- NCP (Network Co Processor) image
- Application Framework
- User Application

Comparing ZigBee® to other connectivity mechanisms, ZigBee® has no kernel driver support. All device driver roles are conducted using the NCP. The ZigBee® software stack uses the EZSP (EmberZNet Serial Protocol) protocol which is an UART protocol that communicates between the NCP and the host process. The NCP image includes both the MAC stack and the ZigBee®/ZigBee® Pro stack. *Figure 11* depicts the ZigBee® architecture.

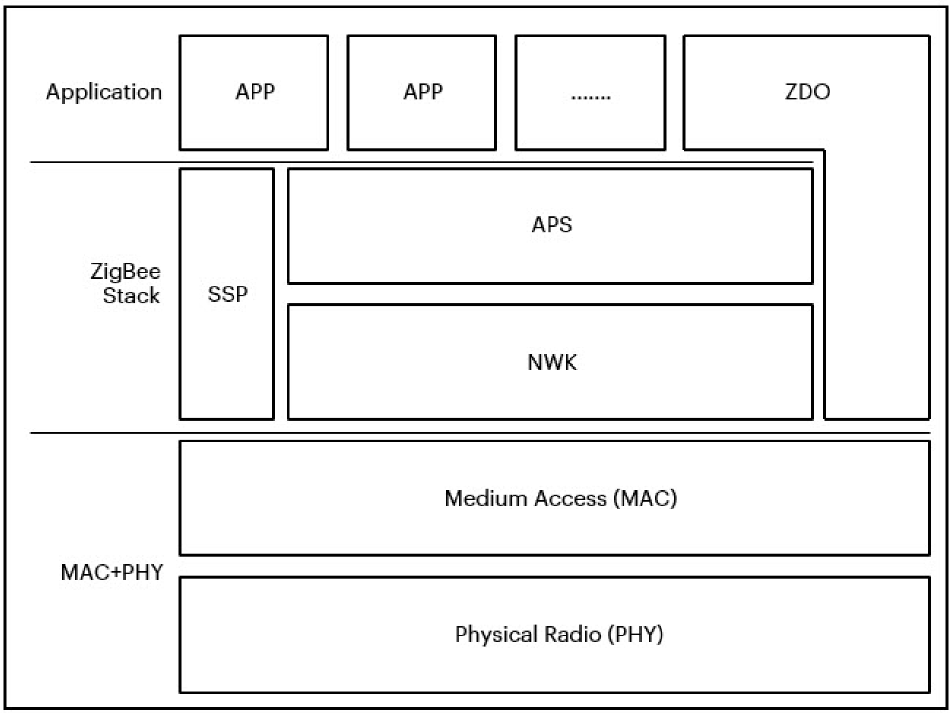

Figure 11. ZigBee® Architecture

The host application includes the application framework that provides functionality such as:

- ZCL (ZigBee® Cluster Library)
- EZSP communication
- User specific application

The user specific application performs control of pre-defined or custom devices adding endpoints and specific roles to these end points using user code in typical callback functions. *Figure 12* depicts the software architecture of the host application.

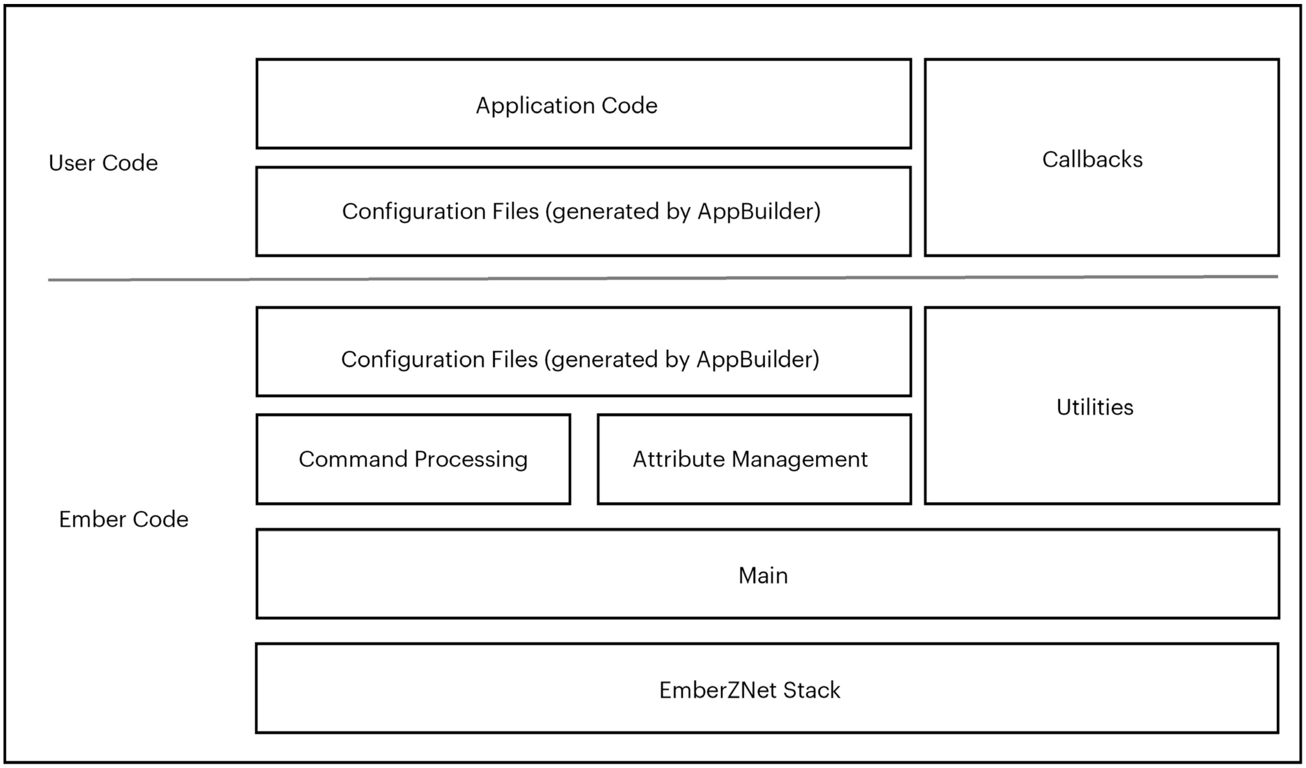

Figure 12. Host Application Architecture

802.11

The 802.11 support that is available on the Hardware Modules consists of a combination of SoC's that have both kernel and user mode driver functionality. The firmware that is available on the various SoC's assure that lower level functionality that requires real time response is taken care of immediately. The firmware interfaces with the kernel driver module "cfg80211" that is the configuration API for 802.11 devices under Linux®. When support for "soft mac drivers" is required, the "mac80211" module is tasked to support such operations. This redistribution of tasks will assure that the SoC firmware can continue to meet real time requirements. The "nl80211" provides the interface between user space and kernel space. User space applications can get access using 802.11 devices via this library. *Figure 13* depicts the architecture required to communicate using the 802.11 user space driver.

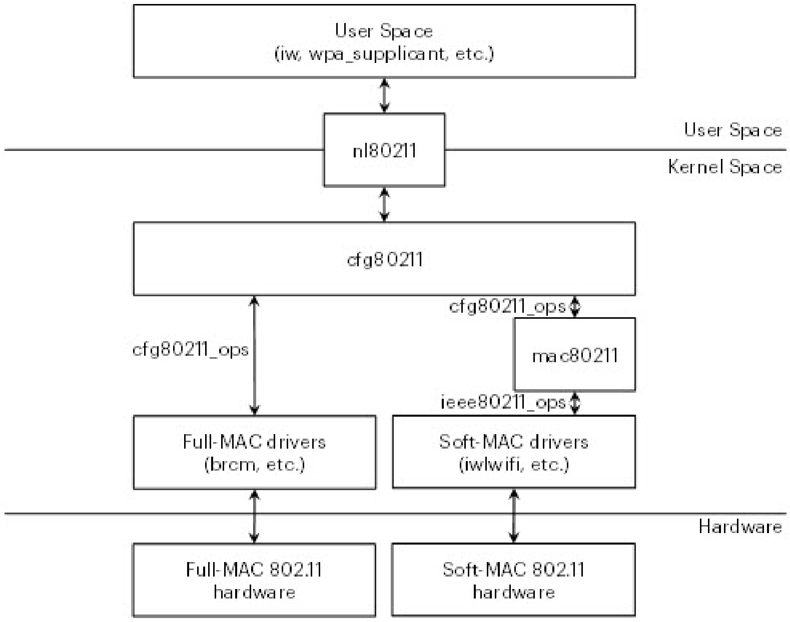

Figure 13. User Kernel Space Communication Mechanism

The various connectivity architectures (Bluetooth®, ZigBee® and 802.11) that are in place for the Hardware Modules are summarized in *Table 2*.

Table 2. Communication Architectures

WPA_SUPLICANT	BlueZ	EmberNet
Nl80211	HCI	EZSP
802.11 + 802.15.1		802.15.4

Network Protocols

The following network protocols that are relevant in IoT space are supported on our Hardware Modules:

- IoTivity using the Open Connectivity Foundation Standard
- CoAP using libcoap
- MQTT using mosquito
- mDNS using Avahi
- LWM2M using wakaama

The Hardware Modules will protect all data that is transmitted using TLS/DTLS mechanism via OpenSSL or other crypto libraries.

IoTivity

IoTivity is an open source software framework enabling seamless device-to-device connectivity to address the emerging needs of the Internet of Things. It is based on an implementation of the Open Connectivity Foundation standard. The IoTivity stack is described in Figure 14.

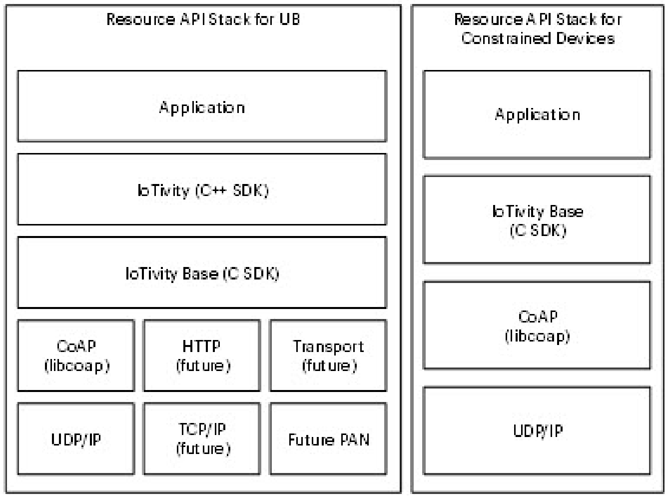

Figure 14. IoTivity Architecture

CoAP

CoAP is developed in the IETF working group "CoRE", libcoap is a C implementation of the CoAP standard. CoAP is similar to HTTP, but it is more light-weight specifically geared towards IoT devices.

MQTT

MQTT is a machine-to-machine (M2M) IoT connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport system. It is useful for connections with remote locations where a small code footprint is required. It has three kinds of roles:

- Publisher: Publish Message to the Subscriber
- Subscriber: Get a Message from the Subscriber
- Broker: Manages both Publisher and Subscriber and relays messages

mDNS

mDNS is a multicast based device discovery service on a local network. It is a "zero-configuration" service, and it enables discovery without a local name server.

LWM2M

LWM2M is a protocol from the Open Mobile Alliance for M2M or IoT device management.

The Network protocols stack implemented on the Hardware Modules is as described in *Figure 15*.

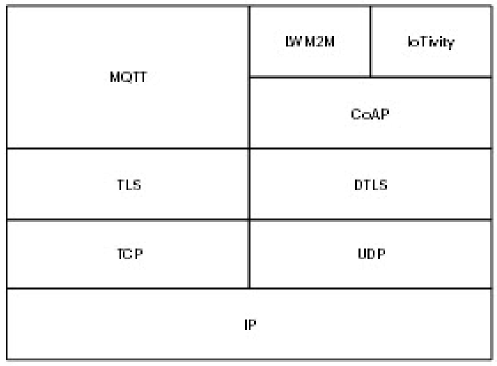

Figure 15. Network Protocol Stack

FIRMWARE UPDATE

eMMC Update/Recovery (fusing) using the SD Card

When developing your own firmware stack you want to be able to update the current firmware present on your Hardware Module. In addition there might be a need to recover or roll back to a previous version of the firmware. In both cases it shows the various steps to follow when a firmware update or recovery is required. The mechanism used by
U-Boot to upgrade the firmware and therefore the partition table is through copying the new image stored on an SD card. See *Figure 16* for details.

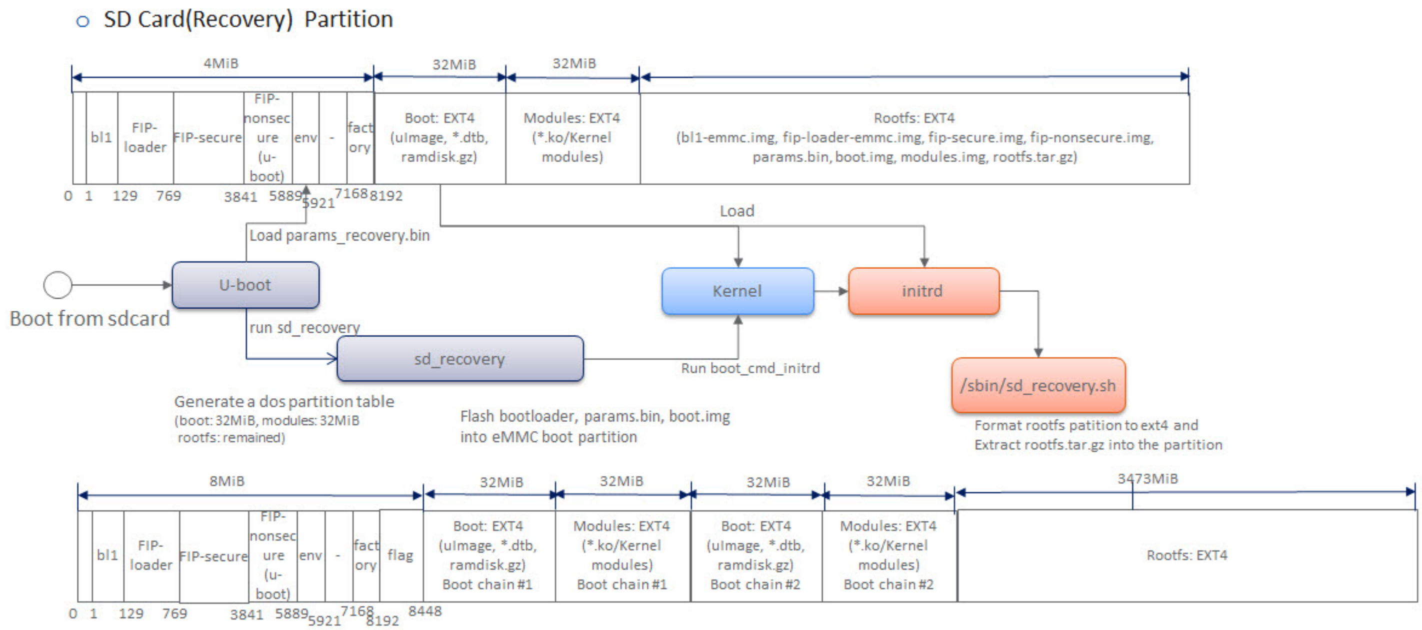

Figure 16. Recovery Sequence Using SD Card Image

U-Boot will flash most of the recovery binaries except for the RootFS system. U-Boot will run the "sd_recovery" command using "partmap_emmc.txt" that contains the partition table. Due to the limited flash size the RootFS image will be extracted from the initial ramdisk.

LEGAL INFORMATION