Application Note

Fast Boot Method for SITARA AM62Ax Family of Processors

章勇，华腾 *Central FAE*

ABSTRACT

This document was translated from a simplified Chinese source. (ZHCAG72)

In automotive and industrial applications, there are many cases where fast boot after power-up is demanded. For example, in an automotive camera monitor system (CMS) or streaming monitor system, the entire process from power-up to display must be completed typically within a few seconds. With the default image used, the boot process of full features often takes more than 20s. The optimization of the boot cycle involves optimizing the boot process, trimming the kernel, and selecting the proper boot media. In this article, the CMS application of the AM62A is used as an example to introduce the fast boot method. In the test described in this article, the duration from power-up to application booting is 2.5s, ideal for the vast majority of applications that demand fast boot.

Table of Contents

List of Figures

Trademarks

1 Introduction

The SITARA AM62Ax family of processors includes the AM62A3 and AM62A7 processors. TI provides several out-of-the-box image files for this chip family, allowing users to quickly burn model image files using a burning tool for development debugging. However, after developing a user application using the default image and deploying the processor in actual products, developers often encounter the issue that the application takes too long to boot after power-on: With the default image used, as well as an SD card used as the boot media, a single boot cycle often takes more than 20 seconds. Such a long boot time is below end users' expectations. To satisfy the need for fast boot, we must modify the boot process, change the boot media, trim the operating system, and so forth.

2 Boot Process

Before learning the boot process, it is important to have a general understanding of the high-level composition of the processor. Most of TI’s processors have multi-core heterogeneous architectures, where multiple compute units of different architectures are present and processed in parallel in a single processor, and some compute units are grouped together to form a domain. Using the AM62A7 as an example, as shown in the following figure, there are up to four A53 cores in the Application Cores domain, typically running a higher-level operating system such as Linux or QNX. The MCU Channel with FFI domain has one standalone MCU R5F core for functional safety, typically running an RTOS or AUTOSAR. One DM R5F core in the Device Management domain is responsible for some jobs in the boot process and for the management of on-chip resources after booting, typically running the RTOS operating system. The Deep Learning Accelerator domain includes one C7xMMA core running the RTOS operating system, which is responsible for AI-powered inference acceleration.

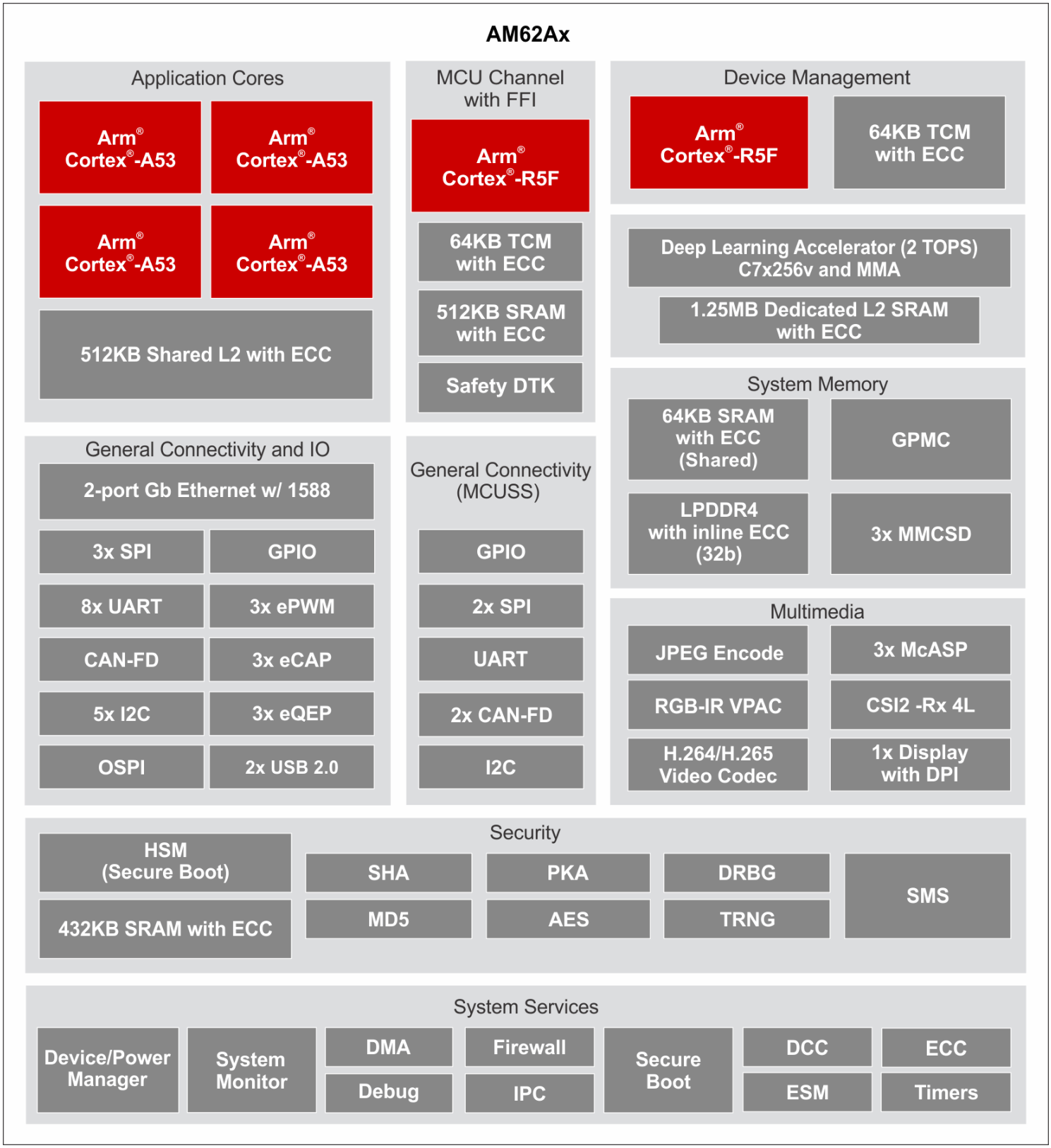

Figure 2-1. AM62Ax System Block Diagram

A complete boot process of the SoC involves loading the respective code for each of the cores. The boot process provided by TI can be divided into SPL and SBL by mode in general, and there are subdivided boot

stages that involve or bypass U-Boot in these two modes. The two boot modes, different configurations available for each mode, and the impact of different configurations are described in detail below.

2.1 SPL Boot Process

As shown in Figure 2-2 and Figure 2-3, the SPL boot process begins with the PMIC power-on, followed by DM R5F starting the ROM code, which loads and boots the R5 SPL code. The compiled and packaged R5 SPL code is named tiboot3.bin. Subsequently, DM R5F loads tispl.bin, and the A-core starts executing the code in tispl.bin, which covers the boot process of TF-A+OPTEE+A53 SPL. Next, the A-core loads u-boot.img and boots the U-Boot operating system. As the U-Boot stage starts, the U-Boot operating system reads the additional configuration for the Device Tree in uEnv.txt. After that, U-Boot reads the Linux kernel program and programs for small cores, such as C7x and MCU R5F, from the file system, booting Linux while simultaneously starting the small core programs. Finally, the Linux kernel finishes booting, followed by the startup of user-mode services, which closes the entire boot process. See also the User Guide of the SDK for the detailed boot process.

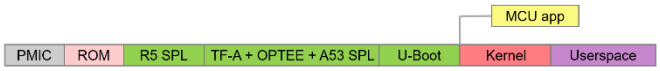

Figure 2-2. SPL Boot Process

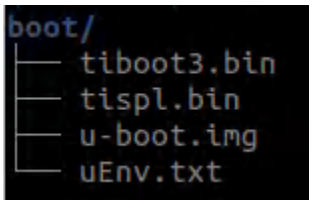

Figure 2-3. BOOT Partition File Structure

This boot process proceeds under the default state. Users can refer to Create SD Card to create a bootable card and perform a test using the SD card boot mode. The ROM code is hardened into the chip before delivery, and the SDK does not provide the ROM code. The sections below describe specific actions of each step in conjunction with the SDK.

2.1.1 R5 SPL

R5 SPL first reads boot information from the ROM, including some basic chip attributes. It then initializes the R5F core, looks for and initializes the serial port to facilitate some boot log output, loads the system firmware, and finally initializes the DDR. Click here to check the specific code, or obtain the code by installing the LINUX SDK.

In the LINUX SDK, the image file tiboot3.bin for R5 SPL is obtained by compiling U-Boot. TI's U-Boot uses Device Tree as the minimum hardware description for boot to ensure that basic peripherals are available. It is configured by am62ax_evm_r5_defconfig. Binman, the packaging tool for the firmware image, manages configurations centrally through the Device Tree, where k3-am62a-sk-binman.dtsi describes the layout of the tiboot3 image. Binman then packages the image according to this layout and groups several components into one or a few files. Notice the corresponding Device Tree and defconfig when modifying R5 SPL.

The following figure illustrates the tiboot3.bin image file packaged by Binman according to the corresponding Device Tree. The package has an X.509 certificate as the header, and the corresponding R5 u-boot-spl.bin as the actual R5 SPL code. Also included in the package is sys-firmware (also known as TIFS), which is loaded to the small core M4F_0 in the on-chip SMS subsystem. The package also contains a minimal Device Tree that describes the minimum device configuration required for boot.

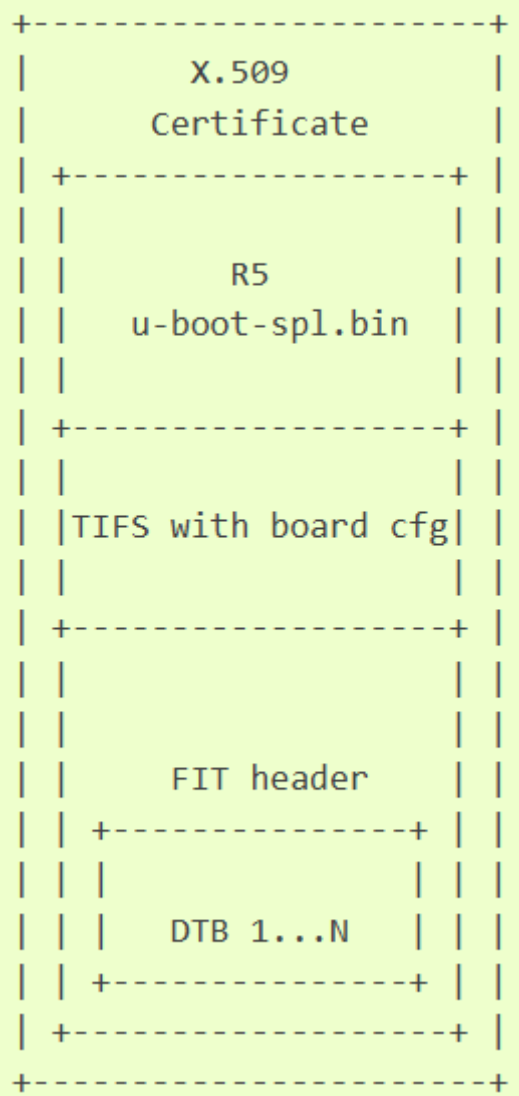

Figure 2-4. tiboot3 Image Package

During compilation, the Binman image packaging process can be shown by setting the environment variables BINMAN_VERBOSE=3 and BINMAN_DEBUG=1. The packaged items recorded in the following Binman log are consistent with those described in the Device Tree.

```
Pack completed after 1 pass(es)
Writing image to './custMpk.pem'
Wrote 0xcab bytes
Pack completed after 1 pass(es)
Writing image to './ti-degenerate-key.pem'
Wrote 0x21a bytes
Pack completed after 1 pass(es)
Writing image to './board-cfg.bin'
Wrote 0x1d bytes
Pack completed after 1 pass(es)
Writing image to './pm-cfg.bin'
Wrote 0x2 bytes
Pack completed after 1 pass(es)
Writing image to './rm-cfg.bin'
Wrote 0x596 bytes
Pack completed after 1 pass(es)
Writing image to './sec-cfg.bin'
Wrote 0x15d bytes
Pack completed after 1 pass(es)
Writing image to './combined-tifs-cfg.bin'
Wrote 0x674 bytes
```

```
Pack completed after 1 pass(es)
Writing image to './combined-dm-cfg.bin'
Wrote 0x5aa bytes
Pack completed after 1 pass(es)
Writing image to './combined-sysfw-cfg.bin'
Wrote 0x734 bytes
Pack completed after 1 pass(es)
Writing image to './tiboot3-am62ax-hs-evm.bin'
Wrote 0x50981 bytes
Pack completed after 1 pass(es)
Writing image to './tiboot3-am62ax-hs-fs-evm.bin'
Wrote 0x50981 bytes
Pack completed after 1 pass(es)
Writing image to './tiboot3-am62ax-gp-evm.bin'
Wrote 0x4ff64 bytes
```

During the boot process, the log for R5 SPL is as follows:

```
[2025-08-05 19:53:27.057] U-Boot SPL 2024.04-ti-g29d0c23d67ee (Nov 29 2024 - 11:41:54 +0000)
[2025-08-05 19:53:27.057] SYSFW ABI: 4.0 (firmware rev 0x000a '10.1.8--v10.01.08 (Fiery Fox)')
[2025-08-05 19:53:27.120] SPL initial stack usage: 13568 bytes
[2025-08-05 19:53:27.120] Trying to boot from MMC2
[2025-08-05 19:53:27.265] Authentication passed
[2025-08-05 19:53:27.265] Authentication passed
[2025-08-05 19:53:27.281] Authentication passed
[2025-08-05 19:53:27.409] Authentication passed
[2025-08-05 19:53:27.409] Authentication passed
[2025-08-05 19:53:27.425] Starting ATF on ARM64 core...
```

The R5 SPL process takes a short time, which mainly comes from data movement from boot media to the SoC's internal RAM and initialization of some resources. The resource initialization stage can hardly be omitted. Therefore, the key to improving the boot speed of this stage lies in using fast boot media.

2.1.2 TF-A+OPTEE+A53 SPL

As the name shows, the TF-A+OPTEE+A53 SPL (hereinafter referred to as A53 SPL) process consists of loading and running the ARM Trusted Firmware (ATF), OP-TEE, and A53 SPL. These images are all packaged in tispl.bin, and configuration descriptions such as the image loading locations in the package are stored in the Device Tree binary k3-binman.dtsi. The structure of tispl.bin is shown in the following figure, including ATF, OP-TEE, DM image, SPL image, and some Device Tree binaries.

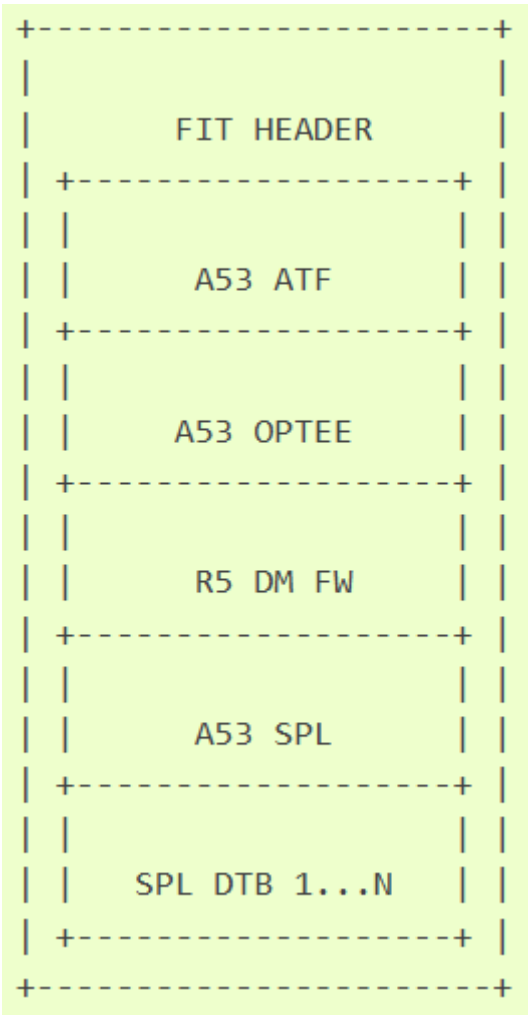

Figure 2-5. tispl Image Package

The corresponding packaging log is shown below. The U-Boot.img is packaged together.

```
Pack completed after 1 pass(es)
Writing image to './custMpk.pem'
Wrote 0xcab bytes
Pack completed after 1 pass(es)
Writing image to './ti-degenerate-key.pem'
Wrote 0x21a bytes
Pack completed after 1 pass(es)
Writing image to './ti-dm.bin'
Wrote 0x8edf0 bytes
Pack completed after 1 pass(es)
Writing image to './tifsstub.bin_hs'
Wrote 0x6d1 bytes
Pack completed after 1 pass(es)
Writing image to './tifsstub.bin_fs'
Wrote 0x0 bytes
Pack completed after 1 pass(es)
Writing image to './tifsstub.bin_gp'
Wrote 0x3ca bytes
Pack completed after 1 pass(es)
Writing image to './tispl.bin'
Wrote 0x165bab bytes
Pack completed after 1 pass(es)
Writing image to './u-boot.img'
Wrote 0x11ff63 bytes
Pack completed after 1 pass(es)
Writing image to './tispl.bin_unsigned'
Wrote 0x163a77 bytes
Pack completed after 1 pass(es)
```

```
Writing image to './u-boot.img_unsigned'
Wrote 0x11f21b bytes
```

The boot log for A53 SPL is as follows. This stage takes approximately 1.4s. The majority of the time is spent on moving and loading the corresponding image files, which is closely related to the boot media.

```
 [2025-08-05 19:53:27.425]
[2025-08-05 19:53:27.425] NOTICE: BL31: v2.11.0(release):v2.11.0-906-g58b25570c9-dirty
[2025-08-05 19:53:27.425] NOTICE: BL31: Built : 04:20:32, Nov 1 2024
[2025-08-05 19:53:27.681]
[2025-08-05 19:53:27.681] U-Boot SPL 2024.04-ti-g29d0c23d67ee (Nov 29 2024 - 11:41:54 +0000)
[2025-08-05 19:53:27.681] SYSFW ABI: 4.0 (firmware rev 0x000a '10.1.8--v10.01.08 (Fiery Fox)')
[2025-08-05 19:53:27.696] Trying to boot from MMC2
 [2025-08-05 19:53:28.193] Authentication passed

 [2025-08-05 19:53:28.833] Authentication passed
```

At this stage, two images also undergo authentication, indicated by the "Authentication passed" message in the boot log, corresponding to the code ti_secure_image_post_process(). The authentication only applies to the DM image and the Device Tree binaries, while ATF and OP-TEE are not authenticated. This process consumes less time and has little impact on the overall boot speed.

2.1.3 U-Boot

At the U-Boot stage, the U-Boot operating system is running, generating the following U-Boot operating system log. At the U-Boot stage, there is a 3s delay to wait for user input on the serial port, which decides whether to proceed with the U-Boot command line. If the command line is accessed, users can enter U-Boot commands to finish early preparations, such as modifying the Device Tree, partitioning the embedded MultiMediaCard (eMMC), and loading images. U-Boot also changes the rate of the boot media and reconfigures the voltage and associated clock according to the supported transfer rate of the boot media. In addition, only at this stage can media such as an SD card be set to high-speed mode, where corresponding image loading is faster. Before being modified to high-speed mode, the boot media typically operates in low-speed mode, with a lower image loading speed. See also the image loading process at the R5 SPL and A53 SPL stages for more information.

```
[2025-08-05 19:53:30.032]
[2025-08-05 19:53:30.032]
[2025-08-05 19:53:30.032] U-Boot 2024.04-ti-g29d0c23d67ee (Nov 29 2024 - 11:41:54 +0000)
[2025-08-05 19:53:30.048]
[2025-08-05 19:53:30.048] SoC: AM62AX SR1.0 HS-FS
[2025-08-05 19:53:30.048] Model: Texas Instruments AM62A7 SK
[2025-08-05 19:53:30.048] DRAM: 2 GiB (effective 4 GiB)
[2025-08-05 19:53:30.544] Core: 87 devices, 30 uclasses, devicetree: separate
[2025-08-05 19:53:30.545] MMC: mmc@fa10000: 0, mmc@fa00000: 1
[2025-08-05 19:53:30.560] Loading Environment from nowhere... OK
[2025-08-05 19:53:30.576] In: serial@2800000
[2025-08-05 19:53:30.576] Out: serial@2800000
[2025-08-05 19:53:30.576] Err: serial@2800000
[2025-08-05 19:53:30.576] Net: eth0: ethernet@8000000port@1
 [2025-08-05 19:53:30.608] Hit any key to stop autoboot: 0
 [2025-08-05 19:53:32.784] switch to partitions #0, OK
[2025-08-05 19:53:32.784] mmc1 is current device
[2025-08-05 19:53:33.056] SD/MMC found on device 1
[2025-08-05 19:53:33.104] 653 bytes read in 35 ms (17.6 KiB/s)
[2025-08-05 19:53:33.104] Loaded env from uEnv.txt
[2025-08-05 19:53:33.104] Importing environment from mmc1 ...
[2025-08-05 19:53:33.168] 53172 bytes read in 51 ms (1017.6 KiB/s)
[2025-08-05 19:53:33.168] Warning: Did not detect image signing certificate. Skipping
authentication to prevent boot failure. This wills
[2025-08-05 19:53:33.184] Load Remote Processor 0 with data@addr=0x82000000 53172 bytes: Success!
[2025-08-05 19:53:33.360] 10719312 bytes read in 159 ms (64.3 MiB/s)
[2025-08-05 19:53:33.360] Warning: Did not detect image signing certificate. Skipping
authentication to prevent boot failure. This wills
[2025-08-05 19:53:33.376] Load Remote Processor 2 with data@addr=0x82000000 10719312 bytes: Success!
[2025-08-05 19:53:33.648] 21029376 bytes read in 256 ms (78.3 MiB/s)
[2025-08-05 19:53:33.696] 61175 bytes read in 37 ms (1.6 MiB/s)
[2025-08-05 19:53:33.696] Working FDT set to 88000000
[2025-08-05 19:53:33.696] ## Flattened Device Tree blob at 88000000
[2025-08-05 19:53:33.712] Booting using the fdt blob at 0x88000000
[2025-08-05 19:53:33.712] Working FDT set to 88000000
```

```
[2025-08-05 19:53:33.744] Loading Device Tree to 000000008feee000, end 000000008fffffff ... OK
[2025-08-05 19:53:33.744] Working FDT set to 8feee000
```

U-Boot also reads the corresponding Device Tree, loads images for small cores, and boots other small cores such as C7. The U-Boot stage takes a long time, typically more than 3s. The time consumption at this stage mainly comes from loading u-boot.img and waiting for user input. When u-boot.img is loaded, the boot media has not switched to high-speed mode. This is why this stage takes a long time. To speed up the U-Boot stage, the 3s delay can be skipped. However, even so, the U-Boot stage still takes more than 1s to complete.

After the U-Boot stage completes, Linux booting begins.

2.1.4 Linux and User Mode

The Linux startup process is relatively conventional. After the Linux kernel initializes, it launches the first program. In the default SDK, we use systemd for resource management and startup of some services. The first one to launch is typically the init program under the sbin/ directory, which loads systemd for file system mounting, peripheral initialization, kernel module loading, network service startup, and so forth. The startup log for the Linux kernel is as follows:

```
[2025-08-05 19:53:33.760]
[2025-08-05 19:53:33.760] Starting kernel ...
[2025-08-05 19:53:33.760]
[2025-08-05 19:53:33.792] [ 0.000000] Booting Linux on physical CPU 0x0000000000 [0x410fd034]
[2025-08-05 19:53:33.792] [ 0.000000] Linux version 6.6.58-ti-01497-ga7758da17c28-dirty (oe-user@oe-
host) (aarch64-oe-linux-gcc (GCC4
 ..........................................
 ..........................................
[2025-08-05 19:53:37.040] [ 2.356567] VFS: Mounted root (ext4 filesystem) on device 179:98.
[2025-08-05 19:53:37.057] [ 2.367111] devtmpfs: mounted
[2025-08-05 19:53:37.057] [ 2.370726] Freeing unused kernel memory: 2496K
```

The Linux kernel takes about 5s to start, which can be reduced by trimming the Linux kernel and reducing log output. To effectively speed up service startup, trim the Linux kernel by disabling the kernel services unnecessary for the target application, and reduce the kernel image size. The log output stage takes up a large portion of the boot time, and eliminating unnecessary log output effectively reduces the duration. See Section 4 for how to trim the operating system and reduce the log output.

After the kernel startup completes, the user mode startup is initiated primarily by the first program, init, with systemd handling subsequent startup and configuration. Its boot log is as follows:

```
[2025-08-05 19:53:37.072] [ 2.375331] Run /sbin/init as init process
[2025-08-05 19:53:37.264] [ 2.575462] systemd[1]: System time before build time, advancing clock.
[2025-08-05 19:53:37.312] [ 2.617899] NET: Registered PF_INET6 protocol family
[2025-08-05 19:53:37.312] [ 2.623708] Segment Routing with IPv6
 ..........................................
 ..........................................
[2025-08-05 19:53:48.479] Arago Project am62axx-evm ttyS2
[2025-08-05 19:53:48.479]
[2025-08-05 19:53:48.479] Arago 2023.10 am62axx-evm ttyS2
[2025-08-05 19:53:48.479]
[2025-08-05 19:53:48.479] am62axx-evm login:
```

The boot time following the takeover of systemd is about 10s. In this case, trimming systemd or using sysVinit to enable necessary services first can be effective in reducing the boot time, which has a significant impact on the optimization of the boot speed. Optimization of this stage is discussed further in Section 4.

2.2 SPL Falcon Mode

Falcon mode refers to packaging the Linux image into tispl.bin so that, during the A53 SPL boot stage, it is loaded directly to boot, thereby skipping the U-Boot stage. There are several implementations of the Falcon mode. For the AM62A, starting from SDK11.01, you can use the Falcon mode by specifying it at the Yocto build stage for quick implementation. With SDK11.01, see the appropriate user manual; for later SDK versions, see the relevant sections in their respective user manuals. For other SDK versions, enable it after simple configuration in the Linux SDK. This article describes how to configure it in the SDK. The same approach also applies to versions later than SDK11.01.

2.2.1 Steps

2.2.1.1 Installing Linux SDK

Search ti.com for the SDK for the chip, such as AM62A, whose SDK can be found at https://www.ti.com/tool/PROCESSOR-SDK-AM62A. Select and download the SDK compatible with the PC's operating system (Windows or Linux).

2.2.1.2 Adding Falcon Script

There is no Falcon script in the default SDK. Please refer to the git patch to add it yourself. This script primarily defines the load address and entry points for some images. For example, the following fields mainly define the target address for Linux kernel code image loading.

```
kernel {
 description = "Linux Kernel (64-bit)";
 data = /incbin/("$1");
 type = "standalone";
 os = "linux";
 arch = "arm64";
 compression = "none";
 load = <0x80080000>; /* Needs to match PRELOADED_BL33_BASE in ATF */
 entry = <0x80080000>;
};
```

2.2.1.3 Compiling Linux Device Tree

Because the core of this boot mode lies in skipping U-Boot, some Device Tree description files originally added via Overlay must be included in the main Device Tree description. In addition, the operating system must be informed of the file system location. This article makes the following changes to the main Device Tree source file to achieve fast boot.

```
--- a/arch/arm64/boot/dts/ti/k3-am62a7-sk.dts
+++ b/arch/arm64/boot/dts/ti/k3-am62a7-sk.dts
@@ -30,6 +30,7 @@ aliases {

 chosen {
 stdout-path = "serial2:115200n8";
+ bootargs = "console=ttyS2,115200n8 earlycon=ns16550a,mmio32,0x02800000 loglevel=4 root=/dev/
mmcblk1p2 rw rootfstype=ext4 rootwait";
 };
```

These changes require the Device Tree binary (DTB) to be recompiled. Under the SDK installation directory, recompile the Device Tree by executing the following command.

```
make linux_dtbs
```

If the Linux kernel is further modified, the Linux image must be recompiled.

2.2.1.4 Creating tispl.bin

Run the following command to generate the .its file.

```
ATF=$IMAGES_DIR/bl31.bin TEE=$IMAGES_DIR/bl32.bin \
 DM=$IMAGES_DIR/ti-dm/am62axx/ipc_echo_testb_mcu1_0_release_strip.xer5f \
 /home/ht/ti-processor-sdk-linux-edgeai-am62a-evm-10_01_00_05/board-support/ti-u-boot-2024.04+git/
scripts/tispl-linux-falcon.sh \
 $IMAGES_DIR/../ti-linux-kernel-6.6.58+git-ti/arch/arm64/boot/Image $IMAGES_DIR/../ti-linux-
kernel-6.6.58+git-ti/arch/arm64/boot/dts/ti/k3-am62a7-sk.dtb > tispl-falcon.its
```

In this command, it is primarily necessary to specify the locations of components in the Falcon binary, such as ATF, TEE, and DM. They must be adapted to the practical circumstances. In the command instance, IMAGES_DIR is the board-support/prebuilt-images folder under the SDK installation directory.

Then run the following command to generate tispl-Falcon.bin

```
mkimage -E -f tispl-falcon.its tispl-falcon.bin
```

Running this command generates tispl-Falcon.bin.

2.2.1.5 Creating a Bootable Card

See also the quick evaluation guide to create a bootable SD card. On the bootable card, the default Boot partition will have the tiboot3.bin, tispl.bin, and u-boot.image files, corresponding to R5SPL, A53SPL, and U-Boot, respectively. Replace tispl.bin with tispl-Falcon.bin generated in the previous step. Then insert the card to boot, and you can see in the log that U-Boot is skipped and the Linux kernel is started directly.

2.3 Replacing SPL With SBL

SBL is another boot mode TI offers, which replaces the traditional SPL boot mode and has a boot process as shown below. After the ROM code is executed, the R5 core boots in two steps, SBL1 and SBL2. The images of other small cores are also loaded to boot during the SBL2 stage. This boot mode can also be combined with Falcon mode, skipping U-Boot to start the Trusted Firmware, OP-TEE, and kernel directly and finally booting the file system through the kernel. The booting of the file system can be further divided into two stages. Only the services that require a fast boot are booted in the first stage, and the remaining file system is booted in the second stage.

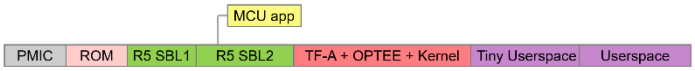

Figure 2-6. SBL Boot Process

To use this boot mode, an additional MCU SDK must be downloaded, which is documented to describe the SBL boot mode under different boot media. This article mainly takes eMMC boot as an example and describes the implementation steps in the eMMC boot case. The impact of boot media lies not only in their transfer rate, but also in the configuration of the high-speed mode in SBL boot mode. This is because the configuration of some boot media is complex, which requires the Linux operating system to be up to complete the high-speed mode configuration. Before the operating system boots, images can only be moved in low-speed mode.

See also the implementation guide for the implementation of SBL, which specifies that the images of the following cores are required:

```
A Linux appimage containing the Linux binaries (ATF, OPTEE, A53 SPL).
tiboot3.bin with SBL stage1, TIFS, BoardConfig
Appimage for SBL stage2
Appimage for MCU R5
Appimage for HSM M4
Appimage for C75
DM firmware appimage for DM R5
```

Among them, the Linux appimage, tiboot3, SBL stage2, and HSM images are described in detail. However, there is a lack of guidance on how to package the appimages for C7 and R5 cores. Therefore, a simple method for packaging these images is introduced here.

Take the C7 core image as an example. After building the MCU Plus SDK, delete the generated appimage, out, and rprc files under the mcu_plus_sdk_am62ax_xxxxxx/examples/drivers/ipc/ipc_rpmsg_echo_linux/am62ax-sk/c75ss0-0_freertos/ti-c7000 directory; copy the out file under the ti-processor-sdk-linux-edgeai-am62a-evm-xxxxx/filesystem/rootfs/lib/firmware/vision_apps_eaik directory in the Linux SDK to the ti-c7000 directory to replace the deleted out file. Note that the names of these two out files must be consistent. Then build the example in the MCU SDK. This is done by packaging the DSP image into the desired appimage using the MCU SDK Build commands.

The R5 image can also be packaged using this method.

After all the required images are packaged, follow the demo running guide to burn them into eMMC, and then boot.

2.3.1 Falcon Boot in SBL

For Falcon boot in SBL mode, configure it primarily by referring to the User Manual for Linux image packaging.

3 Boot Media

Boot media have a significant influence on the boot speed. On the hardware level, the speed of the boot media affects the movement of image files; on the software level, whether the software configures the media to high-speed mode as early as possible also greatly affects the boot speed. In the SDK documentation, the read and write speed test results for the main boot media are listed.

In SPL mode, the configuration of high-speed mode is located in U-Boot. Therefore, enabling Falcon mode to skip U-Boot will cause related drivers to be called to configure the high-speed mode at the subsequent Linux boot stage. High-speed mode can only accelerate the loading of file system-related images, while the Linux Image and other SPL images are still moved in low-speed mode, with loading time exceeding 1s, which is detrimental to fast boot. The following log records the SPL boot in Falcon mode, which takes almost the same amount of time as the regular SPL process, which skips the U-Boot waiting period. In this boot mode, regardless of the boot media, the system runs in low-speed mode until Linux boots. However, it is important to note that this behavior might be modified in later SDK versions. In future versions, the enabling of the high-speed mode might be moved earlier to speed up the boot process. Therefore, users must verify based on the SDK version.

```
[2025-07-04 17:26:03.270] U-Boot SPL 2024.04-ti-g29d0c23d67ee (Nov 29 2024 - 11:41:54 +0000)
[2025-07-04 17:26:03.270] SYSFW ABI: 4.0 (firmware rev 0x000a '10.1.8--v10.01.08 (Fiery Fox)')
[2025-07-04 17:26:03.334] SPL initial stack usage: 13568 bytes
[2025-07-04 17:26:03.334] Trying to boot from MMC2
[2025-07-04 17:26:03.461] Warning: Did not detect image signing certificate. Skipping
authentication to prevent boot failure. This will fail on Security Enforcing(HS-SE) devices
[2025-07-04 17:26:04.229] Warning: Did not detect image signing certificate. Skipping
authentication to prevent boot failure. This will fail on Security Enforcing(HS-SE) devices
[2025-07-04 17:26:04.517] Warning: Did not detect image signing certificate. Skipping
authentication to prevent boot failure. This will fail on Security Enforcing(HS-SE) devices
[2025-07-04 17:26:04.566] Warning: Did not detect image signing certificate. Skipping
authentication to prevent boot failure. This will fail on Security Enforcing(HS-SE) devices
[2025-07-04 17:26:04.566] Warning: Did not detect image signing certificate. Skipping
authentication to prevent boot failure. This will fail on Security Enforcing(HS-SE) devices
[2025-07-04 17:26:04.597] Starting ATF on ARM64 core...
```

In SBL mode, the boot code is more customized, making it easier for users to modify. The SBL boot log in eMMC mode with Falcon is as follows. At the beginning of the boot process, the boot media is set to high-speed mode at the SBL1 stage. As a result, during the subsequent image loading process, image movement in high-speed mode dramatically reduces the boot time. Before Linux boots, the total time consumed is only 600ms. See the following boot log for details.

```
 [2025-07-08 10:03:51.246] SYSFW Firmware revision 0xa
[2025-07-08 10:03:51.246] SYSFW ABI revision 4.0
[2025-07-08 10:03:51.246]
[2025-07-08 10:03:51.262] [BOOTLOADER_PROFILE] Boot Media : eMMC
[2025-07-08 10:03:51.278] [BOOTLOADER_PROFILE] Boot Media Clock : 200.000 MHz
[2025-07-08 10:03:51.278] [BOOTLOADER_PROFILE] Boot Image Size : 163 KB
[2025-07-08 10:03:51.278] [BOOTLOADER_PROFILE] Cores present :
[2025-07-08 10:03:51.294] mcu-r5f0-0
[2025-07-08 10:03:51.294] r5f0-0
[2025-07-08 10:03:51.294] [BOOTLOADER PROFILE] System_init : 36549us
[2025-07-08 10:03:51.294] [BOOTLOADER PROFILE] Board_init : 0us
[2025-07-08 10:03:51.294] [BOOTLOADER PROFILE] Drivers_open : 20614us
[2025-07-08 10:03:51.310] [BOOTLOADER PROFILE] Board_driversOpen : 0us
[2025-07-08 10:03:51.310] [BOOTLOADER PROFILE] Sciclient Get Version : 10174us
[2025-07-08 10:03:51.326] [BOOTLOADER PROFILE] MCU R5 Image Load : 7232us
[2025-07-08 10:03:51.326] [BOOTLOADER PROFILE] DM R5 Image Load : 8510us
[2025-07-08 10:03:51.326] [BOOTLOADER_PROFILE] SBL Total Time Taken : 83081us
[2025-07-08 10:03:51.342]
[2025-07-08 10:03:51.342] Image loading done, switching to application ...
[2025-07-08 10:03:51.342] Starting MCU-r5f and 2nd stage bootloader
[2025-07-08 10:03:51.374]
[2025-07-08 10:03:51.374] SYSFW Firmware Version 10.1.8--v10.01.08 (Fiery Fox)
[2025-07-08 10:03:51.390] SYSFW Firmware revision 0xa
[2025-07-08 10:03:51.390] SYSFW ABI revision 4.0
[2025-07-08 10:03:51.390]
[2025-07-08 10:03:51.758] ERROR: Bootloader_verifyMulticoreImage:684: Failed to authenticate Image
[2025-07-08 10:03:51.758] [BOOTLOADER_PROFILE] Boot Media : eMMC
[2025-07-08 10:03:51.758] [BOOTLOADER_PROFILE] Boot Media Clock : 200.000 MHz
[2025-07-08 10:03:51.774] [BOOTLOADER_PROFILE] Boot Image Size : 16589 KB
[2025-07-08 10:03:51.774] [BOOTLOADER_PROFILE] Cores present :
[2025-07-08 10:03:51.774] hsm-m4f0-0
```

```
[2025-07-08 10:03:51.774] r5f0-0
[2025-07-08 10:03:51.774] a530-0
[2025-07-08 10:03:51.774] [BOOTLOADER PROFILE] System_init : 1645us
[2025-07-08 10:03:51.790] [BOOTLOADER PROFILE] Board_init : 0us
[2025-07-08 10:03:51.790] [BOOTLOADER PROFILE] Drivers_open : 20540us
[2025-07-08 10:03:51.806] [BOOTLOADER PROFILE] Board_driversOpen : 0us
[2025-07-08 10:03:51.806] [BOOTLOADER PROFILE] Sciclient Get Version : 10234us
[2025-07-08 10:03:51.806] [BOOTLOADER PROFILE] HSM Image Load : 5857us
[2025-07-08 10:03:51.822] [BOOTLOADER PROFILE] DM R5 Image Load : 15695us
[2025-07-08 10:03:51.822] [BOOTLOADER PROFILE] A53 Image Load : 291147us
 [2025-07-08 10:03:51.838] [BOOTLOADER_PROFILE] SBL Total Time Taken : 345123us
```

4 Operating System Boot Optimization

4.1 Trimming Operating System

An operating system has two impacts on boot: affecting the image movement speed through the image size and requiring quite a long time to run the initialization code for some kernel modules. Therefore, trimming the operating system helps speed up image movement and kernel startup.

To do so, the main approach is to disable some kernel modules or change them to dynamically loaded ones. This can be controlled by directly modifying the definition in the arch/arm64/configs/defconfig file or using the Linux configuration tool.

4.1.1 Using Preset Trimming Configuration

For fast boot, TI provides an additional overlay config to help trim the kernel. In the kernel code, kernel/configs/ti_early_display.config provides additional kernel module configurations. The following description is included in this configuration and details the boot optimization achieved by disabling the module.

```
#This reduces kernel image size by 0KB and boot time by 144ms
CONFIG_RODATA_FULL_DEFAULT_ENABLED=n
```

To apply the preset fast-boot configuration, the corresponding makerule must be modified. Begin by going to the makerules folder under the Linux SDK installation directory, then modify the Makefile_linux section, and finally add ti_early_display.config to the build command.

```
linux: linux-dtbs
@echo ================================
@echo Building the Linux Kernel
@echo ================================
$(MAKE) -C $(LINUXKERNEL_INSTALL_DIR) ARCH=arm64 CROSS_COMPILE=$(CROSS_COMPILE) defconfig
ti_arm64_prune.config ti_early_display.config
$(MAKE) -j $(MAKE_JOBS) -C $(LINUXKERNEL_INSTALL_DIR) ARCH=arm64 CROSS_COMPILE=$(CROSS_COMPILE)
Image
$(MAKE) -j $(MAKE_JOBS) -C $(LINUXKERNEL_INSTALL_DIR) ARCH=arm64 CROSS_COMPILE=$(CROSS_COMPILE)
modules
ifeq ($(SOC_TYPE),hs)
$(MAKE) linux-fitImage
endif
```

Because Linux relies on the Device Tree, this configuration file must also be added to the build command for the Device Tree. In Makefile_linux-dtbs, add ti_early_display.config to the build command.

```
linux-dtbs:
@echo ================================
@echo Building the Linux Kernel DTBs
@echo ================================
$(MAKE) -C $(LINUXKERNEL_INSTALL_DIR) ARCH=arm64 CROSS_COMPILE=$(CROSS_COMPILE) defconfig
ti_arm64_prune.config ti_early_display.config
@for DTB in $(KERNEL_DEVICETREE); do \
$(MAKE) -j $(MAKE_JOBS) -C $(LINUXKERNEL_INSTALL_DIR) DTC_FLAGS=-@ ARCH=arm64 CROSS_COMPILE=$
(CROSS_COMPILE) $$DTB; \
done
```

After modifying the build command, recompile Linux. The recompiled image file size drops from 20MB to approximately 15MB, and the Linux boot time decreases by approximately 1s.

4.1.2 Custom Trimming

Custom trimming requires modifying the configuration in arch/arm64/configs/defconfig. For example, set CONFIG_PPS to y to compile the code associated with that configuration into the kernel and start it at kernel startup; set the parameter to m to make it a dynamically loaded module, which is loaded when the file system boots after kernel startup is complete; or set the parameter to n to disable it. However, it is important to note that many modules are interdependent, so simply modifying defconfig may not work. The specific effect can be checked after the build by accessing .config under the board-support/ti-linux-xxxx directory under the Linux SDK directory to check the configuration that finally takes effect. For example, although module A is disabled, when module B, which is dependent on module A, is enabled, module A will still be active in .config after the build.

Custom trimming can also be configured using the Linux configuration tool menuconfig, which checks dependencies during configuration to help with efficient configuration.

For example, as the following figure shows, Bluetooth and the associated protocol stack may not be required in the target application. In this case, use menuconfig to open a graphical interface and make a selection with the arrow keys.

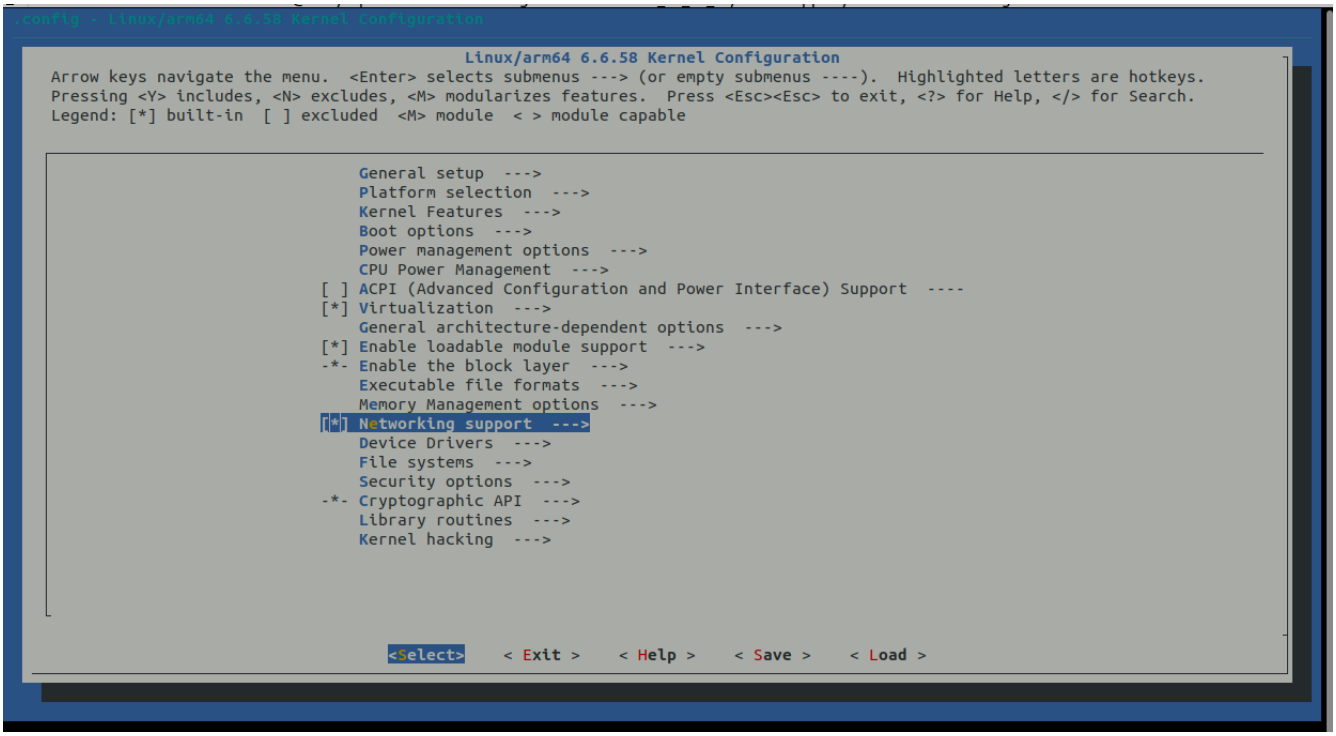

Figure 4-1. Menuconfig Interface

Selecting the Networking support item, you can see a Bluetooth-related option. Select the Bluetooth-related option, and press m to configure the module to be dynamically loaded, press n to disable it, or press y to configure it as a kernel module, which starts with the kernel.

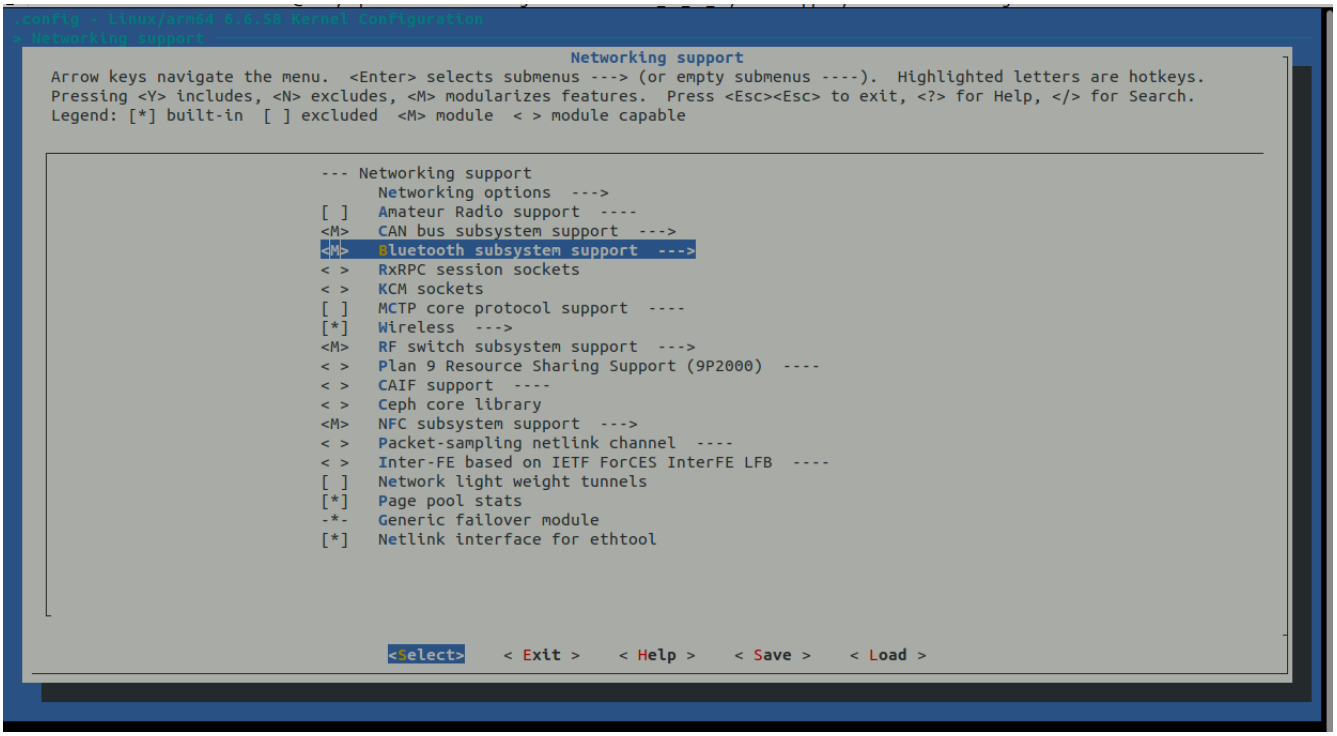

Figure 4-2. Configuring Bluetooth in Menuconfig Interface

When the modifications are complete, select Save.

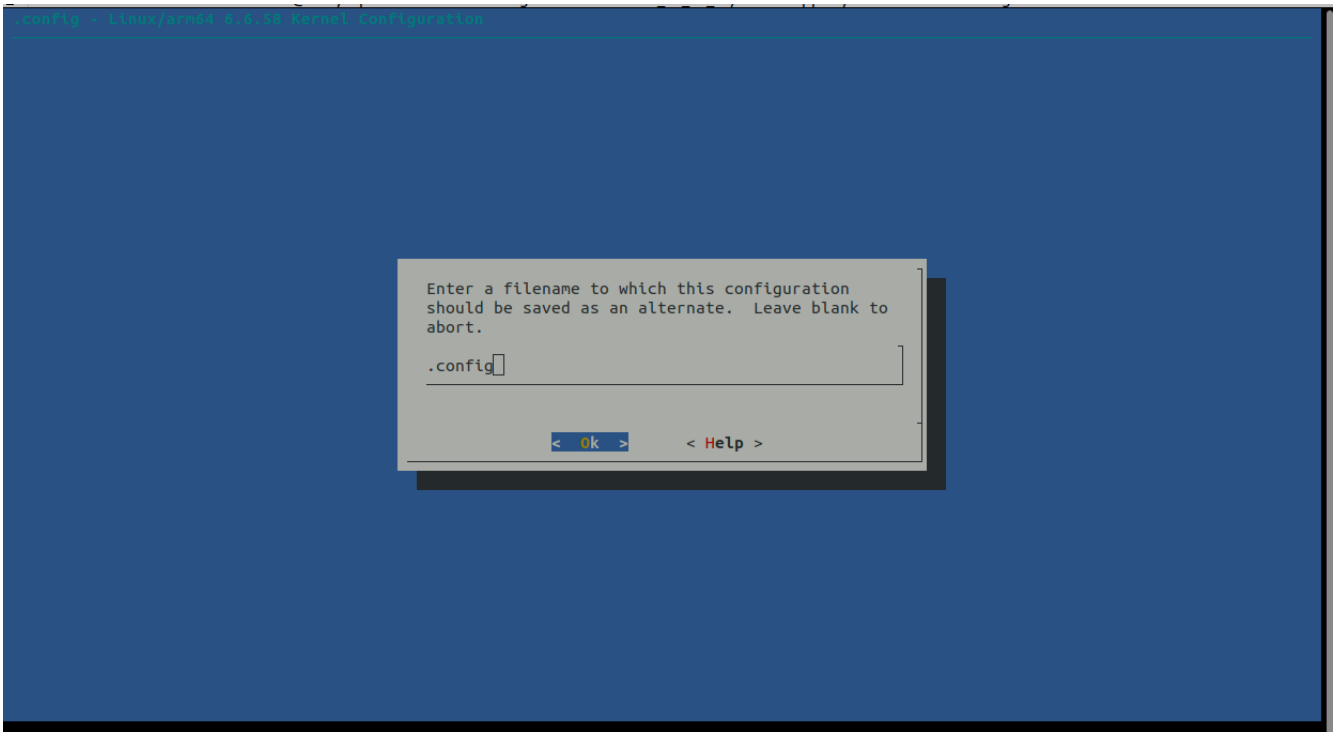

Figure 4-3. Saving Menuconfig

4.2 Reducing Log Output

In the default boot process, the boot log is printed over the serial port during booting. Printing logs greatly increases boot time, while reducing log output can speed up booting by several seconds. See also Adding quiet

to Device Tree for relevant steps. After reducing log output and trimming the kernel, the kernel startup time is reduced to approximately 1s.

5 Userspace Optimization

5.1 Customizing Boot Services Using Systemd

The default SDK file system or userspace services are booted by systemd. The first program that launches after the Linux kernel startup is /sbin/init, which immediately starts the systemd program to complete the remaining configuration. The systemd used by TI conforms to the Linux standard boot process. See also the Linux systemd bootup documentation to learn about the relevant services during boot.

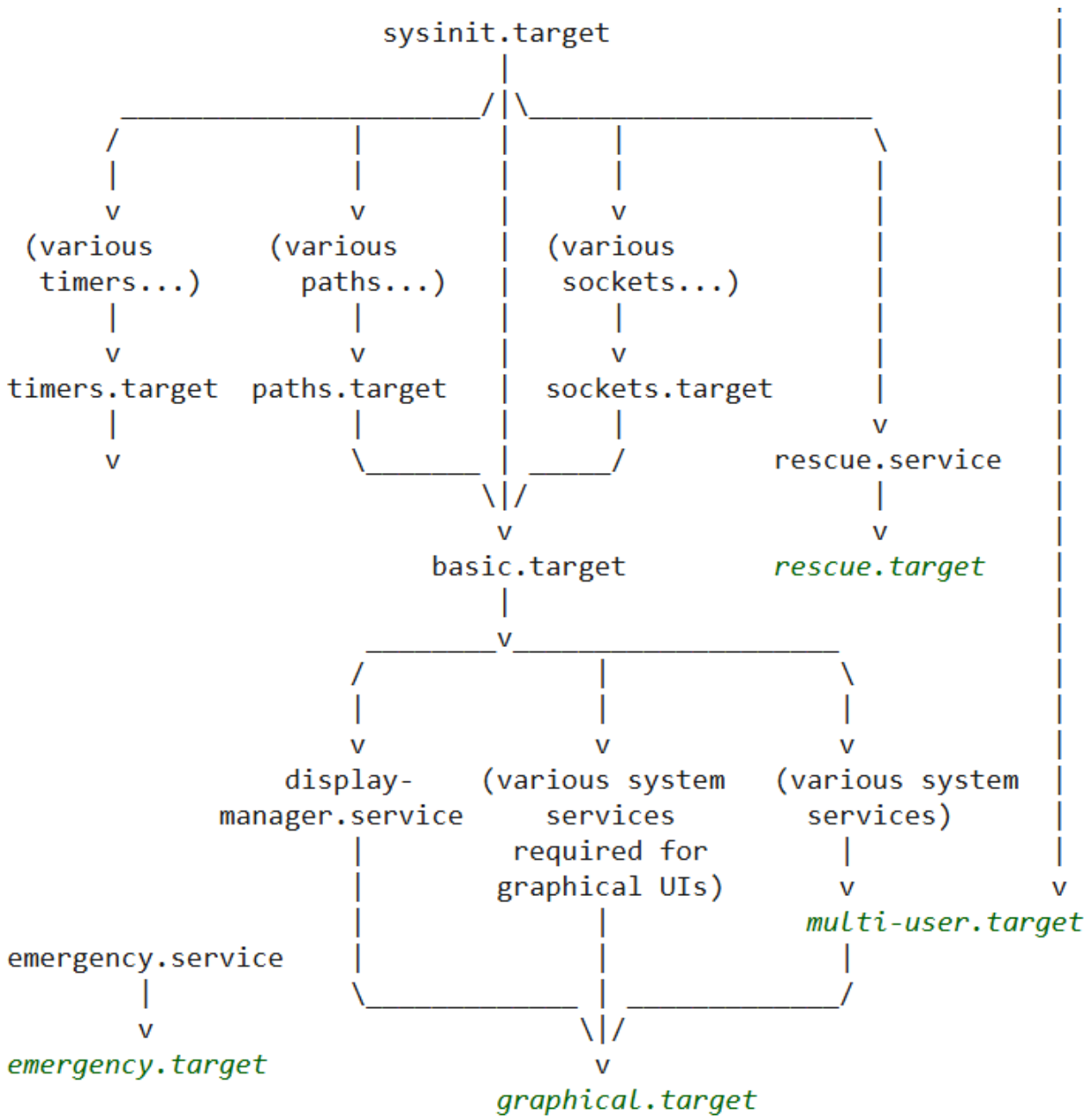

Figure 5-1. Systemd Boot Dependencies

Systemd uses a target for boot service management, and systemd boot finishes when the final target, graphical.target, is running, indicating the end of user-mode booting. The default final target of systemd is graphical.target.

After regular booting, you can access dmesg | grep target to check that the target is set to graphical during systemd booting.

```
systemd[1]: Queued start job for default target Graphical Interface.
```

The status of this target can be viewed using the following command:

```
systemctl status graphical.target
```

After booting is complete, the status is as shown in the log, which indicates that the target has been booted.

```
 am62axx-evm systemd[1]: Reached target Graphical Interface.
```

The following description is also included in the status log, indicating the configuration source for this target:

```
Loaded: loaded (8;;file://am62axx-evm/usr/lib/systemd/system/graphical.target/usr/lib/systemd/
system/graphical.target
```

With this hint, you can view the detailed configuration of this target.

```
# SPDX-License-Identifier: LGPL-2.1-or-later
#
# This file is part of systemd.
#
# systemd is free software; you can redistribute it and/or modify it
# under the terms of the GNU Lesser General Public License as published by
# the Free Software Foundation; either version 2.1 of the License, or
# (at your option) any later version.
[Unit]
Description=Graphical Interface
Documentation=man:systemd.special(7)
Requires=multi-user.target
Wants=display-manager.service
Conflicts=rescue.service rescue.target
After=multi-user.target rescue.service rescue.target display-manager.service
AllowIsolate=yes
```

This target is dependent on units such as multi-user.target. This approach ensures a comprehensive overview of all units currently executed by systemd. After you have a comprehensive overview of all services, you can disable some of them as needed. Use the following command to disable them:

```
systemctl mask <some unit>
```

6 Minimizing Boot Services Using init Script

The first program that starts after the Linux boot is /sbin/init, which can be replaced with a custom boot script. To replace /sbin/init, create a script first, then delete /sbin/init, and finally soft-link the custom script as the boot program using the ln -sf <your script> /sbin/init command.

The following is a reference script for CMS fast boot. This script ultimately launches the edgeai-tiovx-apps-main application. It first executes some mount commands to mount necessary partitions. It then loads several necessary modules using the modprobe command, including camera drivers, csi-rx, v4l2 (for camera data input), rpmsg (for communication with the AI DSP and the R-core that executes the ISP), tidss, and display_connecter (for display). After that, it exports some environment variables to prepare for the final run of services. Finally, it runs the edgeai-tiovx-apps-main executable program.

```
#!/bin/sh
set -x
export PATH=/usr/bin:/sbin:/bin
export LD_LIBRARY_PATH=/lib:/usr/lib:$LD_LIBRARY_PATH
mount -t proc proc /proc
mount -n -t sysfs sysfs /sys
mount -t debugfs debugfs /sys/kernel/debug
mount -t configfs configfs /sys/kernel/config
mount -t efivarfs efivarfs /sys/firmware/efi/efivars
mount -n -t tmpfs none /run
mount -n -t devtmpfs devtmpfs /dev
mkdir -p /dev/shm
chmod 1777 /dev/shm
mount -t tmpfs /dev/shm
export SOC=am62a
export DEVICE_NAME=AM62A
# Load essential kernel modules
insmod /lib/modules/6.6.58-ti-01497-ga7758da17c28-dirty/kernel/drivers/rpmsg/rpmsg_char.ko
insmod /lib/modules/6.6.58-ti-01497-ga7758da17c28-dirty/kernel/drivers/rpmsg/rpmsg_ctrl.ko
modprobe sii902x
modprobe tidss
modprobe display_connector
modprobe ti_k3_r5_remoteproc
modprobe ti_k3_dsp_remoteproc
modprobe omap_mailbox
modprobe cdns_csi2rx
modprobe j721e_csi2rx
modprobe v4l2_cci
modprobe videobuf2_dma_contig
modprobe v4l2_fwnode
modprobe videobuf2_memops
modprobe videobuf2_v4l2
modprobe videobuf2_common
modprobe v4l2_async
modprobe videodev
modprobe cdns_dphy_rx
modprobe mc
modprobe imx219
ulimit -c unlimited
echo "/tmp/core.%e.%p" > /proc/sys/kernel/core_pattern
export EDGEAI_GST_APPS_PATH=/opt/edgeai-gst-apps
export EDGEAI_DATA_PATH=/opt/edgeai-test-data
export OOB_DEMO_ASSETS_PATH=/opt/oob-demo-assets
export MODEL_ZOO_PATH=/opt/model_zoo
export EDGEAI_VERSION=10.1
export EDGEAI_SDK_VERSION=10_01_00
bash /opt/edgeai-gst-apps/scripts/setup_cameras.sh
./opt/edgeai-tiovx-apps/bin/Release/edgeai-tiovx-apps-main ./opt/edgeai-tiovx-apps/configs/linux/
imx219_cam_example.yaml &
echo "System initialized: Hello world!!!!!!"
/bin/sh
```

7 Conclusion

This article describes the fast boot method for the SITARA AM62Ax family of processors. After combining Falcon SBL, eMMC startup, kernel startup optimization, and user mode using the minimal boot script, the AM62A is capable of starting CMS in 2.5s, satisfying the fast-boot needs of applications such as CMS. This article explains in detail how to proceed with each step for quick replication.

8 References

1. https://www.ti.com/product/AM62A7
2. https://www.ti.com/tool/PROCESSOR-SDK-AM62A
3. https://software-dl.ti.com/processor-sdk-linux/esd/AM62AX/11_01_07_05/exports/docs/devices/AM62AX/linux/Linux_Performance_Guide.html
4. https://e2e.ti.com/support/processors-group/processors/f/processors-forum/1510026/am62a7-soc-system-startup-time-is-more-than-20s
5. https://e2e.ti.com/support/processors-group/processors/f/processors-forum/1274738/am62a7-based-on-sdk-9-version-implement-a-cms-project-car-rearview-mirror-how-to-import-ai-tag-function-through-gstreamer
6. https://e2e.ti.com/support/processors-group/processors/f/processors-forum/1479388/am62a7-how-to-set-the-parameter-for-cms-product-what-is-using-for-models-in-pipiline
7. https://e2e.ti.com/support/processors-group/processors/f/processors-forum/1479107/am62a7-cms-resize

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025