Application Note

AM62x 器件上的 Linux 电路板移植

Logan Bristol

摘要

AM62x 片上系统 (SoC) 产品系列专为构建基于 Linux 的成本优化型嵌入式系统而设计。AM62x 具有多种外设和配置，适用于许多市场应用，包括汽车和工业应用。此系列提供许多高级功能，以满足当今系统的需求。对于新用户来说，通过使用这些器件开发定制平台来充分利用这些功能可能会很复杂。本应用手册为 Linux 开发提供了一个清晰的起点，从而提高了基于 AM62x SoC 的定制电路板的开发效率。有关 AM62x 产品的更多信息，请参阅器件特定的数据表、用户指南和 SDK 文档。

内容

表格清单

商标

1 引言

TI 提供了软件开发套件 (SDK) 来加快 AM62x SoC 的开发。该软件包针对评估模块 (EVM) (例如入门套件 (SK)) 进行了调优。SDK 和 SK 旨在针对各种用例快速评估 SoC 的许多功能。所有特性和功能 (例如各种引导模式) 均已启用和优化。虽然这种灵活性对于评估平台非常有用，但为特定用例设计的定制电路板可能不需要增加这种复杂性。例如，在这种情况下，最终产品中只会使用这些引导模式中的几种。

通过 SDK 中提供的完整配置来降低复杂性并不高效，可以将定制电路板启动时间从数天延长到数周。此过程可能涉及随意禁用特性，以确定如何配置更简单的用例。系统的互连性质不利于开发人员找出问题的根源。建议的方法不是剥开复杂系统的各层来找出问题，而是在一开始将最小配置作为坚实的基础，然后迭代到完整和优化的配置。一次逐步添加一项功能可快速识别电路板的哪些方面正常工作，哪些不正常工作。可以针对非功能区域进行调试。

本应用手册详细介绍了如何使用 SDK 添加定制电路板支持。SDK 包含 Linux 内核和 U-Boot (用作引导加载程序) 的源代码库。要启用新的定制电路板，需要将 Linux 和 U-Boot 都移植到定制电路板。首先，为新电路板配置 U-Boot 环境。接下来，开始 U-Boot 电路板移植，首先创建所需的定制板级配置文件并对电路板的新器件树 (一种描述底层硬件的软件结构) 进行初始修改。最后，各个功能和外设会逐渐添加到器件树中，直到完成，并且所有电路板功能都能根据 U-Boot 启动电路板所需正常运行。同一器件树用于在新电路板上启用 Linux 内核，从而完成移植过程。为了解决启动过程中出现的任何错误，本文档还包含调试指南。

建议在收到定制电路板之前，使用 TI EVM 完成本文档中详细介绍的过程。这可在定制电路板到达后实现高效的启动过程，并在已知正常工作的电路板上验证本文档中的步骤。

可以在节 10 中找到软件文件下载链接。虽然本指南和提供的文件是面向 AM62x 系列开发的，但可扩展到节 8 中所详述的其他 TI SoC。

2 安装 SDK

如果尚未在主机上安装 AM62x Linux Processor SDK，则电路板启动的第一步是下载此 SDK。下载说明可在 AM62x Processor SDK 指南中找到。

此 SDK 包含源代码存储库、预构建的二进制文件、文件系统映像和其他有用的开发工具。以下说明基于 AM62x Processor Linux SDK 的 10.x 版本。确切的路径可能因 SDK 版本而异，可能需要搜索目录来确定组件的位置。本文档将 SDK 目录路径称为 TI_SDK。

在 10.x 版本的 SDK 中，U-Boot 存储库位于 *TI_SDK/board-support/ti-u-boot-2024.04+git* 下方。本文档将此路径称为 TI_U_BOOT。

在 10.x 版本的 SDK 中，Linux 内核存储库位于 *TI_SDK/board-support/ti-linux-kernel-6.6.xx+git-ti 下方 (其中 xx 表示将更改的次要版本号)* 。本文档将该路径称为 TI_LINUX。

3 为定制电路板配置 SDK

本节可作为复制定制电路板的 TI U-Boot 基线的分步指南。虽然这些步骤对于开始开发并非绝对必要，但最好每个电路板都使用自己的电路板特定文件，以防止任何冲突或电路板特定的配置问题。创建新的电路板特定文件还会保留随 SDK 提供的可用作参考的 TI EVM 文件。

这些说明将定制电路板命名为 *<boardname>*，将组织命名为 *<company>*。组织名称用于命名目录，以帮助将各自的板级配置文件保存在一起。

1. 复制现有的 EVM A53 和 R5 Kconfig 目标并粘贴到 *TI_U_BOOT/arch/arm/mach-k3/am62x/Kconfig* 中，然后重命名定制电路板的符号。 (可选) 修改相应的电路板特定字符串。下方提供了示例。

```
+ config TARGET_AM625_A53_<BOARDNAME>
+     bool "<COMPANY> K3 based AM625 <BOARDNAME> running on A53"
+     select ARM64
+     select BINMAN
+     select OF_SYSTEM_SETUP

+ config TARGET_AM625_R5_<BOARDNAME>
+     bool "<COMPANY> K3 based AM625 <BOARDNAME> running on R5"
```

```
+    select CPU_V7R
+    select SYS_THUMB_BUILD
+    select K3_LOAD_SYSFW
+    select RAM
+    select SPL_RAM
+    select K3_DDRSS
+    select BINMAN
+    imply SYS_K3_SPL_ATF
```

将以下行添加到同一 Kconfig 文件的底部。

```
+ source "board/<company>/<boardname>/am62x/Kconfig
```

2. 创建新的 <company> 和 <boardname> 目录以存储将要复制和修改的文件。

```
$ mkdir -p TI_U_BOOT/board/<company>/<boardname>/
$ mkdir -p TI_U_BOOT/board/<company>/common/
```

3. 将板级配置文件从 TI 目录复制到新目录，并重命名这些文件。

```
$ cp TI_U_BOOT/board/ti/am62x/* TI_U_BOOT/board/<company>/<boardname>/
$ cp TI_U_BOOT/board/ti/common/* TI_U_BOOT/board/<company>/common/
```

4. 在 *TI_U_BOOT/board/<company>/<boardname>* 目录中，通过修改配置选项的默认值来编辑 Kconfig，如下所示。

```
if TARGET_AM625_A53_<BOARDNAME>
config SYS_BOARD
       default "<boardname>"
config SYS_VENDOR
       default "<company>"
config SYS_CONFIG_NAME
       default "<boardname>_evm"
source "board/<company>/common/Kconfig"
endif

if TARGET_AM625_R5_<BOARDNAME>
config SYS_BOARD
       default "<boardname>"
config SYS_VENDOR
       default "<company>"
config SYS_CONFIG_NAME
       default "<boardname>_evm"
config SPL_LDSCRIPT
    default "arch/arm/mach-omap2/u-boot-spl.lds"
source "board/<company>/common/Kconfig"
endif
```

5. 在同一目录中，将 *evm.c* 文件重命名为 *<boardname>.c*，将 *am62x.env* 文件重命名为 *<boardname>.env*。

```
$ mv TI_U_BOOT/board/<company>/<boardname>/evm.c TI_U_BOOT/board/<company>/<boardname>/
<boardname>.c $ mv TI_U_BOOT/board/<company>/<boardname>/am62x.env TI_U_BOOT/board/<company>/
<boardname>/<boardname>.env
```

6. 在同一目录中，按如下所示编辑 *Makefile*。

```
- obj-y    += evm.o
+ obj-y    += <boardname>.o
```

7. 为定制电路板复制 EVM 电路板头文件并重命名该文件。

```
$ cp TI_U_BOOT/include/configs/am62x_evm.h TI_U_BOOT/include/configs/am62x_<boardname>.h
```

在新创建的头文件中进行以下修改。

```
- #ifndef __CONFIG_AM625_EVM_H
- #define __CONFIG_AM625_EVM_H
+ #ifndef __CONFIG_AM625_<BOARDNAME>_H
+ #define __CONFIG_AM625_<BOARDNAME>_H
```

8. 在 *TI_U_BOOT/configs/* 中创建以下配置片段，并添加以下行以设置 Kconfig 目标。

am62x_<boardname>_r5.config

```
CONFIG_TARGET_AM625_R5_<BOARDNAME>=y # CONFIG_TARGET_AM625_R5_EVM is not set
```

am62x_<boardname>_a53.config

```
CONFIG_TARGET_AM625_A53_<BOARDNAME>=y # CONFIG_TARGET_AM625_A53_EVM is not set
```

您现在已经为定制电路板复制了 TI 的 U-Boot 基线。

4 启动 U-Boot 电路板端口

将 U-Boot 移植到定制电路板主要涉及为电路板创建新的器件树文件。此文件包含一个树状结构，将硬件描述为软件的上层，在本例中为 U-Boot。例如，此文件包含特定于电路板的信息，以使 UART 端口能够用作控制台输出。对整个器件树规格有基本了解将有助于完成此文件并加快移植过程。

4.1 器件树简介

器件树包含表示外设的节点。每个节点都包含一些属性，这些属性保存的数据可供 U-Boot 和 Linux 内核器件驱动程序用于初始化外设。如果没有器件树，则需要在器件驱动程序中对硬件配置和参数进行硬编码。将硬件配置与电路板初始化代码分离可以简化开发和维护过程。

备注

建议在开始开发之前对器件有基本的了解。可在节 10 中找到 Bootlin 创建的一个优质资源。

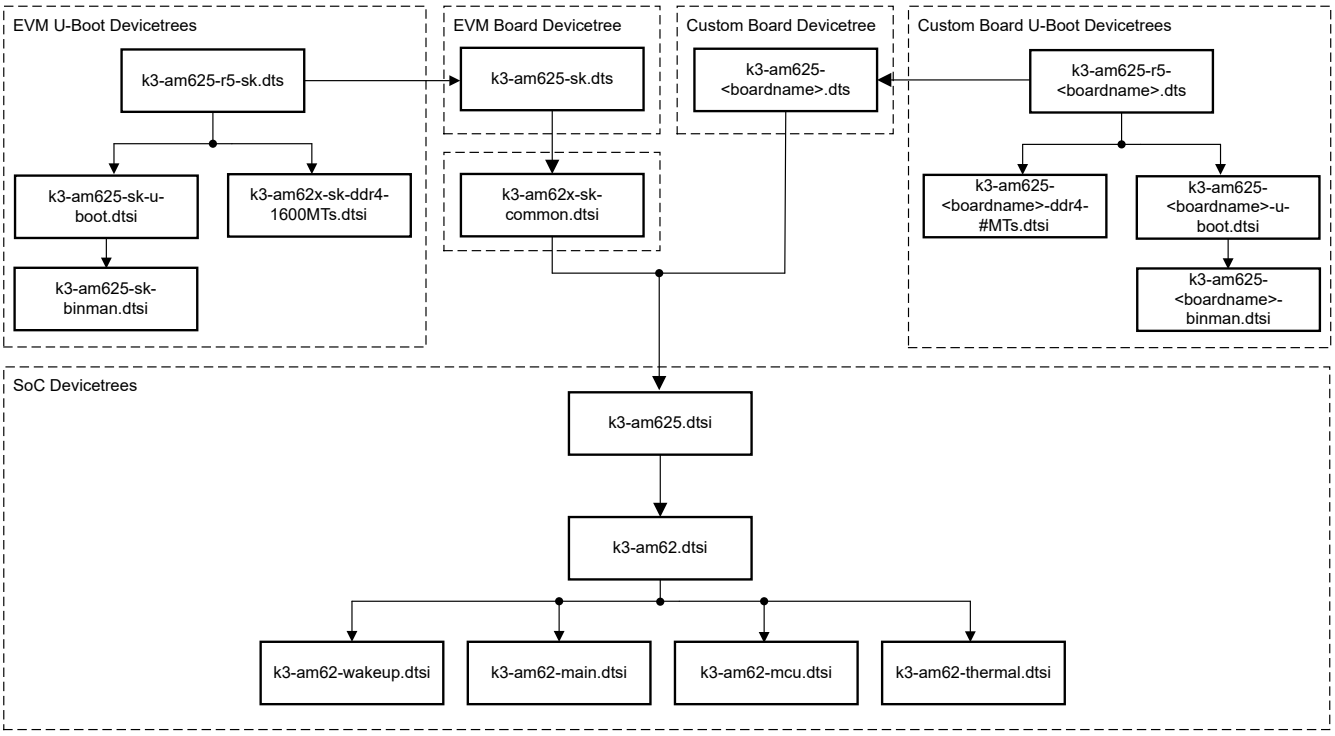

图 4-1. AM62x 器件树结构

了解 AM62x 器件树结构和您将创建的结构的层非常重要。名称中包含 *<boardname>* 的文件是成为定制电路板器件树的文件。

树的基址由 *devicetree include* (DTSI) 文件构成。这些用于定义 SoC 特定的硬件，该硬件在相同 SoC 的所有电路板型号（包括定制电路板）中通常是相同的。直接编辑这些文件可能会破坏包含这些文件的其他器件树。因此，不应直接对其进行修改。相反，应通过引用电路板器件树中的节点来修改 SoC 器件树文件中的器件树节点。

k3-am62x-sk-common.dtsi 文件包含 AM62x SoC 电路板型号之间的通用硬件说明。此 DTSI 包含在 *k3-am625-sk.dts* 和 *k3-am62-lp-sk.dts* 中。文件 *k3-am625-sk.dts* 和 *k3-am62x-sk-common.dtsi* 应用于 SoC DTSI 文件顶部。*k3-am62x-sk-common.dtsi* 不与定制电路板器件树配合使用。

树的顶部有*器件树源* (DTS) 文件。这些文件用于描述特定于电路板的硬件以及修改 DTSI 节点，而不直接更改它们。这就是我们将为定制电路板创建的文件。此外，您还将创建构建引导加载程序二进制文件所需的其他 U-Boot 特定器件。节 4.3 中对此过程进行了介绍。

4.2 最小配置的功能

最小配置由最小器件树和使 U-Boot 和 Linux 内核可以访问这些器件树的配置文件组成。在开始扩展器件树以完全支持定制电路板之前，这种最小配置可用于通过一组较少的外设测试初始电路板功能。为简单起见，最小配置最初支持的外设仅限于 SD 读卡器和 UART。如果定制电路板依赖不同的外设进行引导，或无法使用这些外设进行引导，则需要创建新的器件树节点。

最小配置会减慢速度并禁用 SD 读卡器的高级功能。SD 读卡器被迫使用 3.3V 信号以 25MHz 旧版 SDR 运行，以支持更常见的 SD 卡类别。这种支持减少了与高速或低压功能相关的问题数量，如果电路板不能正确支持这些高级功能，则在尝试启动至 Linux 内核时，此类功能通常会导致故障。出于类似的原因，DDR 存储器也配置为较低的速度。新电路板在以理想速度运行时出现间歇性问题是很常见的，因此取消对这些接口的调优有助于隔离问题并加快进度。

使用此最小配置，可以验证定制电路板的功能，并逐步扩展以包括全系列外设和经过优化的功能。本指南是面向 AM62x 开发的，但可扩展到节 8 中所述的其他 TI SoC。

4.3 准备定制板级配置文件

可根据以下步骤设置用于开发的定制电路板器件树和配置文件。不妨参考 AM62x 器件树图，了解正在创建的器件树如何装入现有结构中。

1. 复制和重命名所需的电路板器件树。

   使用以下命令创建所提供器件树的重命名副本。如果安装的 SDK 中不存在所提供的器件树，则可以在参考部分中下载这些器件树。

   ```
   $ cp TI_U_BOOT/arch/arm/dts/k3-am625-minimal.dts TI_U_BOOT/arch/arm/dts/k3-am625-<boardname>.dts
   $ cp TI_U_BOOT/arch/arm/dts/k3-am625-r5-minimal.dts TI_U_BOOT/arch/arm/dts/k3-am625-r5-
   <boardname>.dts
   $ cp TI_U_BOOT/arch/arm/dts/k3-am625-minimal-u-boot.dtsi TI_U_BOOT/arch/arm/dts/k3-am625-
   <boardname>-u-boot.dtsi
   ```

   这些命名更改需要反映在 *k3-am625-r5-<boardname>.dts* 中的 *#include* 预处理器指令中。

   ```
   // in k3-am625-r5-<boardname>.dts - #include "k3-am625-minimal.dts" - #include "k3-am625-
   minimal-u-boot.dtsi" + #include "k3-am625-<boardname>.dts" + #include "k3-am625-
   <boardname>.dtsi"
   ```

2. 为 DDR 配置生成器件树。

   U-Boot 用于 DDR 初始化的 DDR 配置包含在使用 TI 的 SysConfig 工具生成的器件树中。启动 SysConfig 工具并登录 myTI 帐户后，将“Software Product”字段设置为“DDR Configuration for AM64x, AM625, AM623, AM62Ax, AM62Px”，并将“Device”字段设置为“AM62x”。启动程序以加载 DDR 配置工具。您可以在此处修改 DDR 设置以匹配硬件规格。为防止可能导致引导失败的信号完整性问题，建议将 DDR 频率设置为 667MHz (1334MTs)。这是 DLL 开启时 DDR4 支持的最慢速度。DDR 频率在 SysConfig DDR 工具的“Memory Frequency (MHz)”字段中设置。完成电路板启动后，应根据 DDR 器件规范生成新的 DDR 配置。有关此工具的完整指南，请参阅 SysConfig DDR 配置自述文件。需要包含的生成文件名为 *k3-am62x-ddr-config.dtsi*。下载此文件并使用以下命令重命名此器件树，然后将其放置在 SDK U-Boot 存储库中。

   ```
   $ cp k3-am62x-ddr-config.dtsi TI_U_BOOT/arch/arm/dts/k3-am62x-<boardname>-ddr4-<#MTs>.dtsi
   ```

   替换 *k3-am625-r5-<boardname>.dts* 中的以下行。

   ```
   - #include "k3-am62x-sk-ddr4-1600MTs.dtsi" + #include "k3-am62x-<boardname>-ddr4-<#MTs>.dtsi"
   ```

3. 为定制电路板设置 Binman。

   Binman 用于生成 U-Boot 二进制文件。使用以下命令复制现有的 EVM binman 器件树文件，并为定制电路板重命名该文件。

```
$ cp TI_U_BOOT/arch/arm/dts/k3-am625-sk-binman.dtsi TI_U_BOOT/arch/arm/dts/k3-am625-<boardname>-binman.dtsi
```

   在这个新创建的文件中，进行以下修改以使用自定义器件树。

```
- #define SPL_AM625_SK_DTB "spl/dts/k3-am625-sk.dtb"
+ #define SPL_AM625_SK_DTB "spl/dts/k3-am625-<boardname>.dtb"
```

   将这个新文件包含在 *k3-am625-<boardname>-u-boot.dtsi* 中。

```
- #include "k3-am625-sk-binman.dtsi" + #include "k3-am625-<boardname>-binman.dtsi"
```

4. 将器件树集成到构建流程中。

   通过在 *TI_U_BOOT/arch/arm/dts/Makefile* 中添加以下行，新创建的器件被添加到 *Makefile* 中。

```
dtb-$(CONFIG_SOC_K3_AM625) += k3-am625-sk.dtb \
                  k3-am625-r5-sk.dtb \
                  k3-am62sip-r5-sk.dtb \
                  k3-am625-beagleplay.dtb \
                  k3-am625-r5-beagleplay.dtb \
                  k3-am625-verdin-wifi-dev.dtb \
                  k3-am625-verdin-r5.dtb \
                  k3-am625-phyboard-lyra-rdk.dtb \
                  k3-am625-r5-phycore-som-2gb.dtb \
                  k3-am62-lp-sk.dtb \
                  k3-am62-r5-lp-sk.dtb \
+                  k3-am625-<boardname>.dtb \
+                  k3-am625-r5-<boardname>.dtb
```

5. 设置配置片段。

   创建或修改将应用于现有 AM62x EVM 默认配置文件之上的两个配置片段，以更改 U-Boot 正在访问的器件树。应在 *TI_U_BOOT/configs/* 中创建这些文件。如果您是按照节 3 中的步骤操作，则这些文件已创建好。将下面的行添加到配置片段。

   am62x_<boardname>_r5.config

```
CONFIG_DEFAULT_DEVICE_TREE="k3-am625-r5-<boardname>"
```

   am62x_<boardname>_a53.config

```
CONFIG_DEFAULT_DEVICE_TREE="k3-am625-<boardname>" CONFIG_SPL_OF_LIST="k3-am625-<boardname>"
CONFIG_OF_LIST="k3-am625-<boardname>"
```

现在已经在 U-Boot 中设置了定制电路板配置。

4.4 初始器件树修改

需要修改 U-Boot 存储库中提供的定制电路板器件树，才能使其在定制电路板上运行。本节详细介绍了在尝试启动电路板之前需要进行的更改。

所提供的定制电路板器件树中的当前引脚配置是使用适用于 TI EVM 的 TI SysConfig Pinmux 工具生成的。该应用为 TI SoC 生成引脚配置，并将内部外设引脚连接到 SoC 封装上的外部焊球。由于这些连接的选项有限，因此正确的引脚配置也有限。该工具会了解这些限制，并会针对无效配置提供警告和错误。电路板设计人员应使用该工具针对所需的用例开发有效配置。需要在该工具中对配置进行任何更改，以确保不会引发冲突。这种冲突可能会导致出现电路板问题，需要进行调试和修复，从而可能花费非常宝贵的调试时间。该工具会创建用以将配置导入软件的文件，以便在启动时正确配置器件。这些文件需要从电路板设计人员传递给软件开发人员，以便高效地启动电路板。

小心

电路板设计和软件引脚配置必须保持同步。如果电路板设计或软件的引脚多路复用配置无效，则会导致很难诊断和调试电路板问题。所有与引脚相关的电路板更改都需要使用 SysConfig Pinmux 工具进行验证，并整合到软件中。

最小配置示例中提供的引脚配置适用于 TI 板。如果使用 TI 电路板为 AM62x 器件系列验证此过程，则不需要更改配置。要将自定义电路板器件树中提供的引脚配置替换为定制电路板由 SysConfig Pinmux 工具生成的引脚多路复用 DTSI 中的自定义引脚配置，请删除 *k3-am625-<boardname>.dts* 中的 *&main_pmx* 节点。然后，将引脚多路复用 DTSI 的内容粘贴到 *k3-am625-<boardname>.dts* 中。下方显示了这种情况的一个示例。

```
/* Delete provided pin configuration nodes */
- &main_pmx {
- [... UART pin configuration ...]
- [... SD pin configuration ...]
- };

/* Paste new pin configuration nodes from pinmux DTSI */
+ &main_pmx {
+ [... New pin configurations ...]
+ };
```

也需要在外设的器件树节点中设置外设的新引脚配置。通过修改 *k3-am625-<boardname>.dts* 中该节点中的 pinctrl 属性，可实现此目的。

假设新引脚控制器节点包含 UART0 的以下配置，标记为“uart0-custom_pins_default”。

```
uart0-custom_pins_default: uart0-custom-default-pins {
     pinctrl-single,pins = <
          AM62X_IOPAD(0x01c8, PIN_INPUT, 0) /* (D14) UART0_RXD */
          AM62X_IOPAD(0x01cc, PIN_OUTPUT, 0) /* (E14) UART0_TXD */
     >;
};
```

为了让 UART0 访问这些引脚，请修改 UART0 节点中的 pinctrl 属性，如下所示。确保在“&”后面使用正确的标签，以避免出现器件树编译问题。

```
&main_uart0 {
     bootph-all;
     status = "okay";
     pinctrl-names = "default";
-    pinctrl-0 = <&main_uart0_pins_default>;
+    pinctrl-0 = <&uart0-custom_pins_default>;
};
```

需要对定制电路板上配置的所有外设进行此修改。

可能还需要修改最初在定制电路板器件树中配置的外设实例。定制电路板器件树假设 UART0 用于串行输出和 UART 引导。它还假设 MMC1 用作 SD 读卡器。如果定制电路板上的外设实例不同，则需要修改器件树节点。

在下例中，假设控制台输出的正确 UART 实例是 UART1。按如下所示修改 *k3-am625-<boardname>.dts*，将 UART1 配置为用作控制台输出。

```
- &main_uart0 {
+ &main_uart1 {
      bootph-all;
      status = "okay";
      pinctrl-names = "default";
-     pinctrl-0 = <&uart0-custom_pins_default>;
+     pinctrl-0 = <&uart1-custom_pins_default>;
};
```

如果修改了外设实例，也需要将此实例的引脚配置更改为正确的引脚节点。在以上示例中，pinctrl 属性更改为 UART1 的引脚配置。

如果更改了控制台输出的 UART 实例，则需要修改 *k3-am625-<boardname>.dts* 中的 *aliases* 节点。以下示例演示了如果控制台输出的 UART 实例是 UART1 而非 UART0，则进行此修改。

```
aliases {
-     serial2 = &main_uart0;
+     serial2 = &main_uart1;
};
```

这可确保 Linux 内核使用正确的 UART 实例来输出内核引导日志。

如果定制电路板没有 SD 读卡器，请移除或禁用相应的节点。这可防止发生引导错误。下面显示了此节点被移除的情形。

```
- &sdhci1 {
-       [... Properties ...]
- };
```

在器件树的顶部，兼容与模型属性设置为“minimal”和“ti”。按如下所示为定制电路板修改该字符串。

```
- compatible = "ti,am625-minimal", "ti,am625";
- model = "Texas Instruments AM625 MINIMAL";
+ compatible = "<company>,am625-<boardname>", "<company>,am625";
+ model = "<company> AM625 <boardname>";
```

4.5 构建 U-Boot 二进制文件

本节介绍如何使用自定义电路板配置和 AM62x Processor SDK 创建引导加载程序二进制文件。

对于 AM62x 器件，加载 U-Boot 需要三个映像，这些映像都是由构建过程创建的：

- tiboot3.bin
- tispl.bin
- u-boot.img

电路板上电时，ROM 代码会检查引导模式引脚，并初始化加载 *tiboot3.bin* 并在 32 位 R5 内核上执行它所需的所选引导介质。此映像包含一个称为唤醒 SPL 的二级程序加载程序 (SPL)。此 SPL 会初始化 DDR 和用于加载 *tispl.bin* 的基本电路板元件。此 *tispl.bin* 映像包含在 A53 内核上运行的 64 位主 SPL，以及引导 Linux 所需的固件。主 SPL 会初始化在 *k3-am625-<boardname>.dts* 中配置的外设，用于加载引导流程中的下一个工件。最后，打包在 *u-boot.img* 中的 U-Boot 会正确加载并且可以初始化其他外设，准备引导 Linux 内核组件。

由于启动过程同时涉及 32 位和 64 位内核，因此需要 32 位和 64 位交叉编译器。AM62x Processor SDK 包含这些交叉编译器，但需要为每个编译器设置工具链路径。

设置以下通用环境变量：

表 4-1. 通用环境变量

环境变量	说明
CROSS_COMPILE_32	适用于 Armv7（Arm 32 位）的交叉编译器路径，SDK 提供 arm-oe-eabi-
CROSS_COMPILE_64	适用于 Armv8（Arm 64 位）的交叉编译器路径， SDK 提供 aarch64-oe-linux-
CC_64	Armv8（Arm 64 位）的交叉编译器可执行文件，指定了默认库和头路径，SDK 提供 aarch64-oe-linux-gcc
LNX_FW_PATH	SDK 提供的 TI Linux 固件目录的路径
TFA_PATH	SDK 提供的预编译 Arm Trusted Firmware 的路径 (bl31.bin)
OPTEE_PATH	SDK 提供的预编译 OPTEE 的路径 (bl32.bin)

要设置编译器工具链路径，请参阅 AM62x Processor SDK 指南。以下是有关如何使用 SDK 提供的预编译二进制文件设置固件路径的示例。

```
$ export LNX_FW_PATH=TI_SDK/board-support/prebuilt-images/am62xx-evm/
$ export TFA_PATH=TI_SDK/board-support/prebuilt-images/am62xx-evm/bl31.bin
$ export OPTEE_PATH=TI_SDK/board-support/prebuilt-images/am62xx-evm/bl32.bin
```

设置以下特定于电路板的环境变量：

表 4-2. 特定于电路板的环境变量

环境变量	说明
UBOOT_CFG_CORTEXR	Cortex-R 的默认配置文件
UBOOT_CFG_CORTEXA	Cortex-A 的默认配置文件

设置默认配置变量，以应用先前在现有 EVM 默认配置基础上创建的配置片段。下面是设置这些环境变量的示例。

```
$ export UBOOT_CFG_CORTEXR="am62x_evm_r5_defconfig am62x_<boardname>_r5.config"
$ export UBOOT_CFG_CORTEXA="am62x_evm_a53_defconfig am62x_<boardname>_a53.config"
```

由于构建了 32 位和 64 位二进制文件，因此它们需要输出到每个架构的单独文件夹中。在主机上创建此输出目录。这些指令将输出目录路径称为 OUTPUT_DIR。32 位和 64 位二进制文件内置在 OUTPUT_DIR 的子目录中。在以下指令中，这些子目录被命名为 r5 和 a53，分别用于 32 位和 64 位构建。所有构建命令都从 SDK U-Boot 存储库的根目录下运行。

构建 tiboot3.bin

```
$ make clean O=OUTPUT_DIR/r5
$ make ARCH=arm CROSS_COMPILE="$CROSS_COMPILE_32" $UBOOT_CFG_CORTEXR O=OUTPUT_DIR/r5
$ make ARCH=arm CROSS_COMPILE="$CROSS_COMPILE_32" O=OUTPUT_DIR/r5 BINMAN_INDIRS=$LNX_FW_PATH
```

构建 tispl.bin 和 u-boot.img

```
$ make clean O=OUTPUT_DIR/a53
$ make ARCH=arm CROSS_COMPILE="$CROSS_COMPILE_64" $UBOOT_CFG_CORTEXA O=OUTPUT_DIR/a53
```

```
$ make ARCH=arm CROSS_COMPILE="$CROSS_COMPILE_64" CC="$CC_64" O=OUTPUT_DIR/a53 BL31=$TFA_PATH
TEE=$OPTEE_PATH BINMAN_INDIRS=$LINUX_FW_PATH
```

根据定制电路板上器件的安全级别，使用以下二进制文件。

表 4-3. 根据器件安全级别生成的二进制文件

安全性	生成的二进制文件
GP	tiboot3-am62x-gp-evm.bin tispl.bin_unsigned u-boot.img_unsigned
HS-FS	tiboot3-am62x-hs-fs-evm.bin tispl.bin u-boot.img
HS-SE	tiboot3-am62x-hs-evm.bin tispl.bin u-boot.img

备注

新电路板的器件安全级别可能是 HS-FS。

生成的三个二进制文件位于 OUTPUT_DIR 的 r5 和 a53 子目录的根目录下。这些二进制文件必须确切地重命名为 *tiboot3.bin*、*tispl.bin* 和 *u-boot.img*。这可以使用以下命令来实现。

```
// in the r5 subdirectory of OUTPUT_DIR
$ mv tiboot3-am62x-{gp/hs-fs/hs}.bin tiboot3.bin
// in the a53 subdirectory of OUTPUT_DIR
$ mv tispl.bin{_unsigned} tispl.bin
$ mv u-boot.img{_unsigned} u-boot.img
```

4.6 U-Boot 部署说明

要使用定制电路板配置启动器件，需通过 UART 将上一步中生成的三个引导二进制文件加载到电路板。此方法使用一个简单的通用外设进入 U-Boot 命令提示符。定制电路板需要能够在器件上配置引导模式引脚，以获得有效的 UART 引导设置。TI EVM 上提供了用于更改引导模式设置的开关，以支持各种引导源。在成品板上，这种更改可能通过简单的跳线或其他设置来实现。以下步骤假设电路板已针对 UART 引导进行适当配置。

第一步是在主机 PC 上安装串行通信程序，例如 minicom。配置此程序以正确地与电路板通信。AM62x 快速入门指南包含有关设置此串行连接的说明。在搜索引导映像时，检查是否存在从 ROM 持续输出的“C”字符，以验证是否已建立终端设置和硬件连接。示例输出如下所示。

```
CCCCCCCC
```

如果未看到终端输出，则可能导致此问题的原因有很多。使用已知正常的 TI 电路板验证此步骤会非常有用，可帮助隔离电路板和主机设置之间的问题。必须在继续启动过程之前解决这些问题。

- 电路板问题 (电源、引导模式引脚设置、布线连续性等)
- UART 实例
- UART 引脚配置
- 终端设置

在主机 PC 与电路板之间建立串行连接后，通过 UART 加载引导二进制文件。发送二进制文件的正确顺序是 *tiboot3.bin*、*tispl.bin*、*u-boot.img*。如果使用 minicom 或类似的串行通信程序，则使用内置应用程序发送二进制文件。否则，可以使用命令提示符和 *lrzsz* 包将 UART 二进制文件发送到电路板。有关使用 UART 加载 U-Boot 二进制文件的完整说明，请参阅有关 UART 引导的 AM62x Processor SDK 指南。如该文档所述，XMODEM 协议用于发送 *tiboot3.bin*，YMODEM 协议用于发送 *tispl.bin* 和 *u-boot.img*。

在传输每个二进制文件后，控制台会输出一个文本块。如果成功传输了 *tiboot3.bin* 和 *tispl.bin*，则每个输出的最后一行如下所示。

```
Trying to boot from UART
CCCCC
```

传输 *u-boot.img* 后，器件准备引导 Linux。此时，U-Boot 可让您按任意键退出自动引导。使用此功能可在 U-Boot 提示符处停止引导过程。下面显示了成功进入 U-Boot 命令 shell 的引导日志示例。

```
U-Boot 2024.04-00006-g49c04dedb6d-dirty (Jul 19 2024 - 14:01:17 -0500)

SoC:   AM62X SR1.0 HS-FS
Model: Texas Instruments AM625 MINIMAL
EEPROM not available at 0x50, trying to read at 0x51
Reading on-board EEPROM at 0x51 failed -19
DRAM:  2 GiB
Core:  37 devices, 21 uclasses, devicetree: separate
MMC:   mmc@fa00000: 0
Loading Environment from nowhere... OK
In:    serial
Out:   serial
Err:   serial
Failed to probe am65_cpsw_nuss driver
EEPROM not available at 0x50, trying to read at 0x51
Reading on-board EEPROM at 0x51 failed -19
Net:   No ethernet found.
Hit any key to stop autoboot:  0
=>
```

进入 U-Boot 命令 shell 意味着定制电路板的 UART 器件树配置正常工作。还表明，取消调优的 DDR 设置足以使 U-Boot 正常启动并运行，这是一个非常好的迹象。下一步是引导 Linux 并进入内核命令 shell。这需要 Linux 内核映像、内核器件树和 *initramfs* 文件系统。由于无法通过 UART 有效地传输这么多数据，因此需要使用另一个外设来加载这些映像。

5 扩展定制电路板器件树

此时，基本电路板功能和 UART 配置已通过成功加载 U-Boot 进行了验证。要加载更大的 Linux，需要更快的外设或块存储器件。配置该外设是迭代启动过程的下一步。以以太网为例，本节将演示如何添加新的外设。以太网是需要配置的复杂外设，可作为一个很好的示例来了解配置其他器件树外设节点可能需要用到的器件树资源。如果 SD、eMMC 或 USB 等外设在定制电路板上提供，那么开始使用这些外设会更简单。最小配置中包含 SD 配置示例，如果定制板具有 SD 功能，该配置可能已经正常工作。

5.1 器件树配置

器件树绑定用作有关如何为给定外设创建节点的指南。它们是特定于其所代表的硬件的标准化节点格式。器件树绑定指定如何命名节点以及外设节点所需的属性。在将驱动程序提交给上游维护人员以包含在 U-Boot 和 Linux 等项目中时，需要提供这些绑定。这些文件通常还具有示例节点。使用相同驱动程序的其他器件树也是有关如何为给定外设正确创建新节点的不错示例。

可以在 *TI_LINUX/Documentation/devicetree/bindings/* 中的 Linux 内核文档中找到器件树绑定。U-Boot 也为某些节点提供绑定。可以在 *TI_U_BOOT/include/dt-bindings/* 中找到这些绑定。U-Boot 中的电路板器件树和 Linux 内核是相同的，因此查询这两个源来构建外设节点非常有用。通过对与节点、属性名称和属性设置关联的兼容字符串进行 grep 搜索，可以找到相关的绑定文件。有关使用器件树绑定的更多指导，请参阅节 10 中的 Bootlin 器件树培训。

配置节点的第一步是检查节点的 SoC 器件树定义。SoC 器件树中的节点包含外设所需的大多数配置设置，但由于这些节点不完整，因此将其禁用。这些节点经过修改以匹配电路板的硬件配置，并在电路板器件树文件（本例中为 *k3-am625-<boardname>.dts*）中启用。不妨参考 AM62x 器件树图了解电路板 DTS 文件和 SoC DTSI 文件的交互方式，这可能会有所帮助。节点通常引用位于整合的所有器件树文件中的其他节点。要启用节点，请确保其引用的依赖关系也已启用。研究以下以太网示例应该有助于更清楚地说明这一点。

Cpsw3g 是一种以太网 MAC，在主域中运行。这表示 SoC 器件树节点定义位于 *k3-am62-main.dtsi* 中。此节点引用其他节点，例如 *&k3_clks*、*&main_pktdma* 和 *&phy_gmii_sel*。有必要确保电路板器件树已启用这些节点。通过设置“status”属性启用节点，如下所示。

```
+ &cpsw3g {
+     status = "okay";
+ };
```

除了启用节点外，每个配置的外设都需要将其 pinctrl 属性设置为已粘贴到 *k3-am625-<boardname>.dts* 的正确引脚配置。如需进行此修改，请参阅节 4.4。执行这些步骤后，将在下面设置引脚配置属性。

```
&cpsw3g {
     status = "okay";
     pinctrl-names = "default";
     pinctrl-0 = <rgmii1-custom_pins_default>;
};
```

有关为给定外设包括哪些引脚配置的指导，请参阅 *k3-am625-sk.dts* 和 *k3-am62x-sk-common.dtsi* 中的外设配置。

添加属性“pinctrl-names”并将其设置为“default”，以指定要始终使用的新增引脚配置。某些配置可能需要多个引脚配置。有关配置引脚设置的完整指南，请参阅 *TI_LINUX/Documentation/devicetree/bindings/pinctrl/pinctrl-bindings.txt*。

5.2 描述节点中的外设

一个有助于理解器件树结构的有用工具是反向编译从构建过程生成的器件树二进制文件 (DTB)。这将生成一个可读文件，其中包含已纳入编译过程的器件树上的所有节点。生成的文件便于查看描述外设的所有节点和属性。DTB 位于 *OUTPUT_DIR/a53/k3-am625-<boardname>.dtb*。要对 DTB 进行反向编译，请使用以下命令：

```
$ dtc -I dtb -O dts k3-am625-<boardname>.dtb -o <boardname>-reversed.dts
```

反向编译的 DTB 是 *<boardname>-reversed.dts* 文件。可以将此输出与绑定文件和其他 DTS 文件中的示例进行比较，以确保完整性和正确性。

构建器件树的另一个优质资源是使用其他器件树作为初始化外设的指南。通过编译另一个器件树（例如 *k3-am625-sk.dts*）并反向编译生成的 DTB，可以很容易看到其他外设节点配置。通常，EVM 器件树节点可用作表示定制电路板外设的良好参考。如果外设的使用与 EVM 类似，则可能只需要稍作修改（如果有）。

在定制电路板上进行器件树开发的一种好方法是搜索 EVM 器件树文件 *k3-am62x-sk-common.dtsi* 和 *k3-am625-sk.dts*，以查看在节点的 SoC 定义顶部应用了哪些属性。通常还需要在定制电路板的器件树中设置这些属性。在器件树绑定目录中搜索属性和兼容字符串会生成有关每个属性定义的内容和如何设置该内容的文档。

在 *k3-am625-sk.dts* 中，以太网 PHY 定义为 MDIO 总线节点的子节点。这已添加到下面的电路板器件树中。假设已设置引脚配置和状态属性。

```
cpsw3g_mdio { status = "okay"; pinctrl-names = "default"; pinctrl-0 = <mdio-custom_pins_default>; +
cpsw3g_phy0: ethernet-phy@0 { + }; };
```

在 *k3-am625-sk.dts* 中，在以太网 PHY 节点中定义了一些属性。对内核绑定目录中的任何属性进行 grep 搜索会将我们指向 *ti,dp83867.txt*，这表示 PHY 节点有四个必需属性。这些是 EVM 电路板器件树中指定的相同属性。本文档还将我们指向一个头文件，该文件包含这些属性的不同设置，这些属性是以太网 PHY 的控制寄存器。可以根据 EVM 器件树设置来设置这些设置，也可以通过参考器件规格的以太网端口设计文档来设置。如果使用不同的 PHY，这些值和整合到器件树中的方法将会发生变化。如下图所示：

```
cpsw3g_mdio {
     status = "okay";
     pinctrl-names = "default";
     pinctrl-0 = <mdio-custom_pins_default>;
          cpsw3g_phy0: ethernet-phy@0 {
+              reg = <0>;
```

```
+                ti,rx-internal-delay = <DP83867_RGMIIDCTL_2_25_NS>;
+                ti,tx-internal-delay = <DP83867_RGMIIDCTL_2_75_NS>;
+                ti,fifo-depth = <DP83867_PHYCR_FIFO_DEPTH_4_B_NIB>;
+        };
};
```

EVM 器件树为 *&cpsw_port1* 添加了“phy-handle”属性，该属性将 PHY 节点分配给端口节点。启用该节点后，可以在下面设置此属性。

```
+ &cpsw_port1 { + status = "okay"; + phy-handle = <&cpsw3g_phy0>; + };
```

5.3 修改器件树配置

假设使用此器件树来生成 U-Boot 二进制文件以测试以太网配置。根据上面的章节通过 UART 传输 *u-boot.img* 后，当使用任意键来停止自动引导时，U-Boot 控制台输出将显示在下面。

```
U-Boot 2024.04-00007-g344db2cf625-dirty (Jul 23 2024 - 09:05:14 -0500)

SoC:   AM62X SR1.0 HS-FS
Model: Texas Instruments AM625 SK
EEPROM not available at 0x50, trying to read at 0x51
Reading on-board EEPROM at 0x51 failed -121
DRAM:  2 GiB
Core:  40 devices, 22 uclasses, devicetree: separate
MMC:   mmc@fa10000: 0, mmc@fa00000: 1
Loading Environment from nowhere... OK
In:    serial
Out:   serial
Err:   serial
EEPROM not available at 0x50, trying to read at 0x51
Net:   am65_cpsw_nuss_port ethernet@8000000port@1: Invalid PHY mode, port 1
No ethernet found.

Hit any key to stop autoboot:  0
=>
```

“Net”字段指示此时以太网配置的状态。一条错误消息显示以太网端口 1 将 PHY 识别为在无效模式下运行。这可能指向器件树问题，因为该消息指示 PHY 未在正确的模式下运行，或者以太网端口未识别 PHY 所处的运行模式。

要修复此错误，请研究 PHY 和以太网端口节点是否为可能的原因。在 EVM 器件树中，端口节点具有一个设置为“rgmii-rxid”的属性“phy-mode”。搜索属性赋值“rgmii-rxid”可找到绑定文件 *ethernet-controller.yaml*。此文件指定在端口节点中设置“phy-mode”以指定 PHY 和以太网端口之间的接口。这些绑定指定：如果 PHY 提供 RX 延迟，则“rgmii_rxid”应设置为“phy-mode”。由于 RX 延迟是根据 PHY 绑定在前面步骤中的要求设置的，因此会应用该属性并将其添加到器件树中。

```
&cpsw_port1 { phy-handle = <&cpsw3g_phy0>; + phy-mode = "rgmii-rxid"; };
```

这突出显示了最终生成正常运行的电路板器件树的迭代过程。应用此修复后，通过 UART 重新构建和重新加载 U-Boot 应该会显示一个可用于 U-Boot 的功能正常的以太网端口。现在，该端口可用于传输加载 Linux 所需的较大映像。

6 引导 Linux 内核

6.1 内核引导概述

为了简化启动过程，下一节中的 Linux 内核启动过程使用 SDK 中提供的组件。测试新配置时只需重建器件树组件。内核映像和文件系统在 SDK 中提供，并在整个电路板启动过程中重复使用。此功能凸显了将器件树作为硬件描述语言的价值，因为相同的内核和文件系统可由不同的电路板使用，而无需更改代码。

在上一节中配置的 U-Boot 器件树会重复用作内核器件树。这是因为应该同步开发电路板的 U-Boot 和内核器件树，以简化调试过程。下一节中的说明详细介绍了如何重复使用器件树。

initramfs 文件系统用于在电路板启动期间引导内核。这是为了减少装载物理根文件系统所需的任何硬件依赖性。电路板启动过程完成后，可使用非易失性存储中的物理文件系统（如 eMMC）或通过 NFS 安装的开发人员友好型文件系统来启用完整功能。

6.2 内核部署说明

本节介绍了如何在配置完能够将 Linux 内核映像、器件树和文件系统载入 RAM 的外设后，立即加载这些组件。要测试内核配置，请通过 UART 加载 U-Boot 二进制文件，并按照前面几节中的提示退出自动引导。

此方法使用 AM62x Processor SDK 提供的内核映像和 *initramfs* 文件系统。下表指定了这些映像在 SDK 中的位置。

表 6-1. Linux 内核组件的 SDK 路径

组件	路径
内核映像	TI_SDK/board-support/prebuilt-images/am62xx-evm/Image
Ramdisk	TI_SDK/filesystem/am62xx-evm/tisdk-tiny-initramfs-am62xx-evm.rootfs.cpio

需要编译内核器件树。虽然到目前为止，此器件树将与用于 U-Boot 的器件树完全相同，但必须在 Linux 内核存储库中编译它。

按照以下命令将编辑后的电路板器件树文件从 U-Boot 复制到 Linux 内核存储库并生成内核 DTB。

```
// copy board DTS from U-Boot to kernel
$ cp TI_U_BOOT/arch/arm/dts/k3-am625-<boardname>.dts TI_LINUX/arch/arm64/boot/dts/ti/

// from the root of TI_LINUX
$ make ARCH=arm64 CROSS_COMPILE="$CROSS_COMPILE_64" distclean
$ make ARCH=arm64 CROSS_COMPILE="$CROSS_COMPILE_64" defconfig ti_arm64_prune.config
$ make DTC_FLAGS=-@ ARCH=arm64 CROSS_COMPILE="$CROSS_COMPILE_64" ti/k3-am625-<boardname>.dtb
```

生成的 DTB 可以在 *TI_LINUX/arch/arm64/boot/dts/ti/k3-am625-<boardname>.dtb* 中找到。

内核映像、器件树和文件系统通过外设单独载入 RAM，或从非易失性存储复制。此过程取决于上述章节中在定制电路板上启用的外设或接口。需要将 Linux 映像复制到用于通过可用接口将内核组件转移到器件的外部介质或主机目录。在上面的部分中讨论了以太网，如果以太网正常工作，则可以使用以太网。在以下步骤中，使用 SD 卡是因为 TI 板上已有现成可用的 SD 卡，并且在电路板启动的这一阶段很容易使用。

要使用 SD 卡加载内核组件，需要分区 SD 卡。创建分区 SD 卡的一种方法是使用 SDK 提供的默认映像刷写 SD 卡。这是使用像 *balenaEtcher* 这样的应用程序完成的。默认映像位于 AM62x Processor SDK 下载页面。选择“Downloads”，然后选择 PROCESSOR-SDK-LINUX-AM62X 对应的“Download options”。下载 AM62x Yocto SD 卡映像并将其刷写到 SD 卡中。这将正确地对卡进行分区，并在卡上放置可正常工作版本的 SDK 文件系统。此时不会使用此文件系统。

为了避免混淆，请从引导分区中删除所有文件，并将内核组件复制到该位置。AM62x Processor SDK 指南中提供了 SD 卡刷写指南。

使用 UART 引导以进入 U-Boot 提示符并停止自动引导。对于用于将内核组件载入 RAM 中的外设，请使用 U-Boot 加载命令。请参阅 U-Boot 文档以确定为各个不同的外设使用哪些命令。以下示例说明了如何使用 SD 卡加载所需的内核组件。

```
/* using MMC device 0 (SD), partition 1 */
=> load mmc 0:1 $loadaddr Image
=> load mmc 0:1 $fdtaddr k3-am625-<boardname>.dtb
=> load mmc 0:1 $rdaddr tisdk-tiny-initramfs-am62xx-evm.rootfs.cpio
```

备注

如果此时未设置 *loadaddr*、*fdtaddr* 和 *rdaddr* U-Boot 环境变量，则未为自定义设置正确配置 U-Boot。返回到节 3 更正定制电路板的 U-Boot 环境。

如果映像加载失败，则器件配置可能存在问题。返回到节 5.1 以尝试重新配置器件。

现在，所需的内核组件已经加载到器件 RAM 中，下一步是指定内核初始化过程。对于主板启动，这是输入内核命令 shell 并使用 *initramfs* 作为文件系统。为此，请设置 *bootargs* 环境变量，该变量在 Linux 内核开始引导时从 U-Boot 传递到 Linux 内核。以下 U-Boot 命令用于设置此初始化过程。

```
=> setenv bootargs rdinit=/bin/sh
```

该器件现已准备好尝试使用复制到 RAM 的映像来引导内核。下面的命令会引导存储在 RAM 中的内核组件。

```
=> booti $loadaddr $rdaddr:0x$filesize $fdtaddr
```

Linux 内核将尝试引导。如果成功，内核引导日志将填充初始化消息，并最终进入内核命令 shell。为了访问系统和硬件信息，请使用以下命令装载以下 psuedo-filesystems。

```
# mount -t proc none /proc
# mount -t sysfs none /sys
# mount -t devtmpfs none /dev
```

如果您无法进入内核命令 shell，有多种方法可以调试此问题。有关如何尝试正确重新配置外设以成功引导内核的方法，请参阅节 5.1。有关调试的帮助，请参阅节 7。

进入内核命令行可让您访问在构建器件树和调试引导故障时非常有用的工具。务必继续采用相同的迭代方法，即逐个启用每个外设，并在建立有效的配置后将代码提交到 U-Boot 存储库。节 5.1 中提供的器件树配置方法以及下一节中的工具对于构建器件树非常有用，可用于初始化电路板上的所有外设。

完成电路板启动过程的常见后续步骤包括配置定制电路板以使用外部文件系统、启用高速数据功能以及自定义内核映像。有关更多详细信息，请参阅 AM62x Processor SDK 指南。

7 工具和调试

当尝试启动定制电路板时，预计在启动过程中会遇到问题。本节介绍了启动定制电路板时出现的常见问题以及如何处理调试过程。

7.1 内核调试跟踪

在电路板启动过程的早期，如果内核在输出引导日志之前发生故障，则很难检测引导故障的根本原因。此外，内核引导日志可能难以解读。通过 *menuconfig* 选择向内核映像添加内核调试功能，有助于找出失败的原因。

要启用更多调试功能，需要构建自定义内核映像。这些步骤出自 AM62x Processor SDK 指南。这些命令从 SDK Linux 内核存储库的根目录下运行。

1. 安装使用以下命令构建内核映像所需的工具。

   ```
   sudo apt install git xz-utils build-essential flex bison bc libssl-dev libncurses-dev
   ```

2. 清理源以开始全新构建内核。

   ```
   make ARCH=arm64 CROSS_COMPILE="$CROSS_COMPILE_64" distclean
   ```

3. 进入配置菜单。

   要启用额外的调试功能，请启用额外的配置选项。可使用 *menuconfig* 完成此操作。要对建议的配置文件片段应用 *menuconfig* 设置，请使用以下命令。

   ```
   make ARCH=arm64 CROSS_COMPILE="$CROSS_COMPILE_64" defconfig ti_arm64_prune.config menuconfig
   ```

4. 启用调试功能。

导航至“Kernel hacking”以查看内核调试选项。在 *menuconfig* 中启用调试功能，将这些功能设置为内置到内核映像中。有关这些调试配置选项的说明，请参阅 *TI_LINUX/lib/Kconfig.debug*。启用调试功能后，保存配置并退出配置菜单。使用以下命令构建内核映像。

```
make ARCH=arm64 CROSS_COMPILE="$CROSS_COMPILE_64"
```

生成的映像可在 *TI_Linux/arch/arm64/boot/Image* 中找到。使用此映像可以访问添加的调试功能。

7.2 OpenOCD 调试

建议使用 JTAG 调试电路板以找出问题的根源。OpenOCD 是一款开源调试器，可与各种基于 gdb 的调试 IDE 交互。OpenOCD 使用指南可在 U-Boot 文档中的 K3 Generation 下找到。

8 未来的工作

这种最小配置是为 AM62x 开发的，所使用的概念更广泛地适用于其他 TI SoC，甚至是其他运行 U-Boot 和/或 Linux 的 Arm® 处理器。该配置专门可扩展到 K3 系列中的其他器件（AM64x、AM67x 等）。建议以这种最小配置为例来为其他 SoC 构建类似的器件树。以下步骤可用作创建最小配置的一般方法。

1. 为 U-Boot 和 Linux 内核创建最小器件树

```
k3-{SOC}-minimal.dts k3-{SOC}-r5-minimal.dts k3-{SOC}-minimal-u-boot.dtsi
```

2. 创建最小配置片段以包含最小器件树

```
{SOC}_minimal_a53.config {SOC}_minimal_r5.config
```

9 总结

通过减少已启用外设的数量并降级其功能，可为 SoC 创建最小配置以作为启动定制电路板的初始测试。本应用手册概述了使用最小配置作为设置定制电路板配置的起点的分步过程，旨在简化电路板启动过程。最小配置可简化此过程，并将初始启动过程的时间从数天或数周缩短为数小时。

10 参考资料

- 德州仪器 (TI)，*AM62x 处理器器件版本 1.0 技术参考手册*。
- 德州仪器 (TI)，*适用于 AM62x 的 Processor SDK Linux*。
- Bootlin，器件树入门。
- U-Boot 文档。
- OpenOCD 用户指南。
- 相关软件文件：
  - k3-am625-minimal.dts
  - k3-am625-r5-minimal.dts
  - k3-am625-minimal-u-boot.dtsi

修订历史记录

Changes from Revision * (August 2023) to Revision A (September 2024) **Page**

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265