*Application Note*

# *无闪烁切换的早期启动屏幕*

*Aparna Patra, Devarsh Thakkar, Nikhil Jain, Soumya Tripathy* *Sitara MPU*

**摘要**

本应用说明提供了如何在 AM62P 上尽可能快速地显示启动画面，并实现从启动画面到完整系统用户界面 (UI) 的无故障过渡的相关信息。本文档中的步骤通过使用通用引导加载程序 (U-Boot) 流程以及自定义的 Slim Bootloader (SBL) 和 Linux® 内核进行演示。

## 内容



## 插图清单



## 表格清单



## 商标

# 1 引言

越来越多的汽车、工业和机器人应用场景要求在引导周期中尽早启用显示功能。操作系统涉及多个引导阶段，引导加载程序是在引导至整个系统之前启动并初始化系统的第一个软件组件。本应用报告解释了如何在引导加载程序阶段启用显示，如何在该阶段显示启动画面或动画，并实现无闪烁切换到系统 UI。启动画面启用过程在本报告中进行了说明，涵盖了广泛使用的开源引导加载程序，以及基于 FreeRTOS® 的次级启动加载程序，后者能够通过 Falcon 模式引导流程实现快速启动到 Linux。本指南接着解释了如何在系统引导过程中保持启动画面显示上下文，并实现无闪烁切换到系统 UI。

# 2 所用硬件

## 2.1 AM62Px 处理器

AM62Px（P 表示加强版）是现有 Sitara™ AM62x 低成本系列应用处理器的扩展，专为高性能嵌入式 3D 显示应用而构建。可扩展的 Arm® Cortex®-A53 性能和嵌入式功能（例如多屏幕高清显示支持、3D 图形加速、4K 视频加速和广泛的外设）使 AM62Px 非常适合广泛的汽车和工业应用，包括汽车数字仪表、汽车显示器、工业人机界面 (HMI) 等。

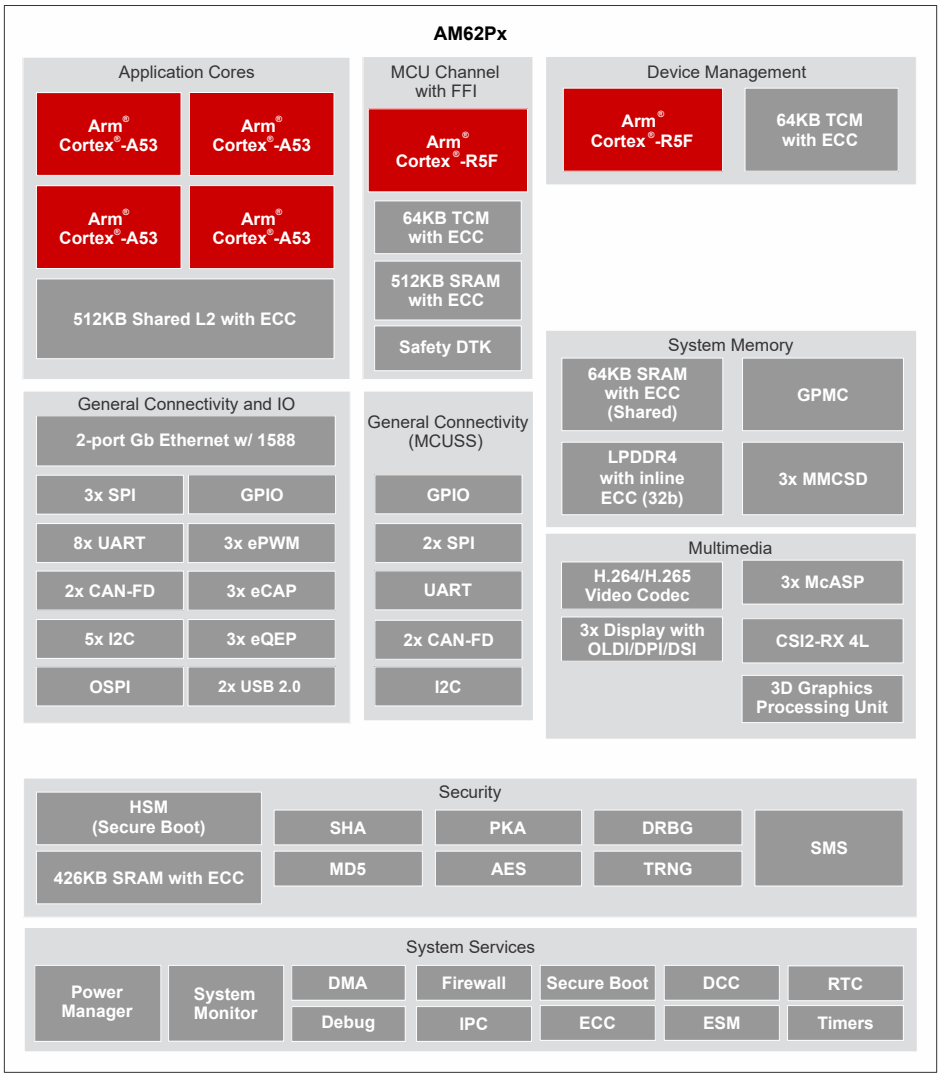


**图 2-1. AM62P 方框图**

## 2.2 SK-LCD1

1920 × 1200 的 Open LVDS 显示接口 (OLDI) 显示器或低电压差分信号 (LVDS) 液晶显示 (LCD) 套件是入门套件 AM62x 处理器评估模块 (EVM) 的附加配件，用于为 HMI、工业计算机以及其他需要显示功能的应用场景提供触摸和显示功能。此型号由薄膜晶体管 (TFT) LCD 面板、驱动电路、背光系统和投射式电容式触控面板组成。

## 2.3 AM62P 的显示子系统

显示子系统 (DSS) 是灵活的多流水线子系统，支持高分辨率显示输出。DSS 包括输入流水线，提供具有透明度的多层混合，以实现动态合成。支持各种像素处理功能，例如颜色空间转换和缩放等。DSS 包括一个直接存储器存取 (DMA) 引擎，允许直接访问帧缓冲区（器件系统内存）。显示输出可以无缝连接到开放式 LVDS 显示接口发送器 (OLDITX)，或者可以作为显示并行接口 (DPI) 直接驱动器件焊盘。

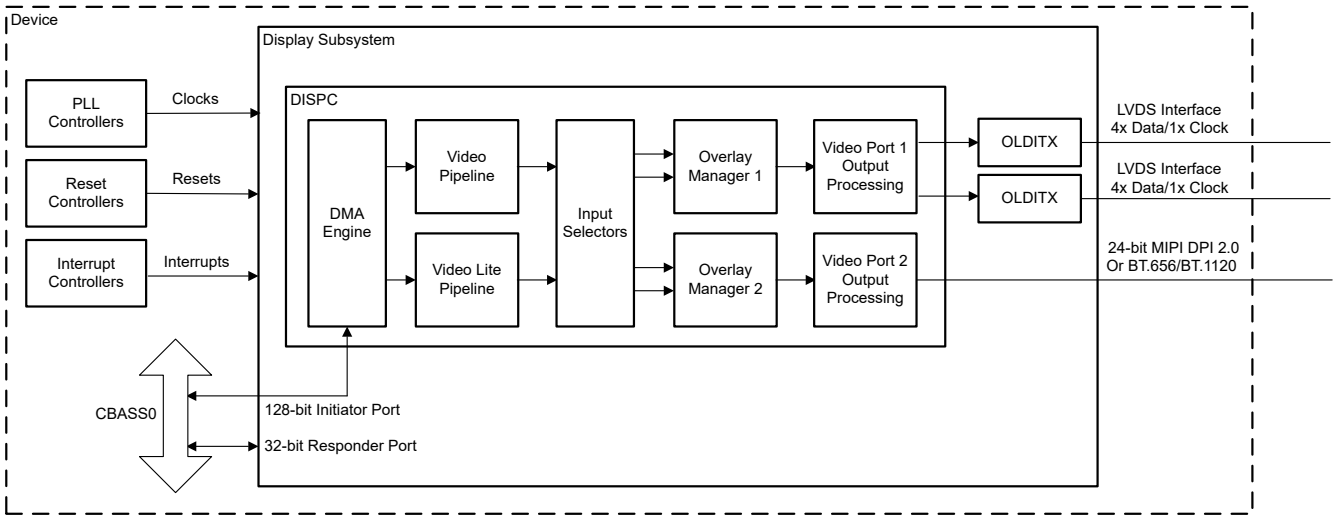


**图 2-2. DSS 方框图**

# 3 早期启动屏幕架构

## 3.1 AM62P 的引导阶段

本节描述了 AM62P 使用的两个引导流程序列。只读存储器 (ROM) 代码是在器件启动时或上电复位 (POR) 后自动运行的第一个代码块。ROM 引导加载程序代码是硬编码到器件中的。由于内部存储器容量有限，这是一段非常小的二进制代码。ROM 引导加载程序启动后，接下来会加载*次级程序* 加载器，以实现 Linux 特定的引导流程，或者加载*次级引导加载程序*，以实现基于实时操作系统 (RTOS) 的引导流程。还可以根据所需用例自定义这些引导序列。


**图 3-1. U-Boot**

1. *次级程序* 加载程序主要用于初始化外部双倍数据速率 (DDR) 存储器，并为下一阶段的引导加载程序 U-Boot 设置引导过程。U-Boot 运行在 DDR 中，提供更广泛的功能，例如命令行 (CMD) 支持、器件驱动程序架构和 Kconfig 架构。然后，U-Boot 加载内核镜像，以便在 A53 内核上启动高层操作系统 (HLOS)，如 Linux。

2. *次级引导加载程序* 是基于 FreeRTOS 的引导加载程序，负责执行特定于器件的初始化，加载相应的二进制文件以初始化后续内核，并最终启动应用程序。在 AM62P 上，*次级引导加载程序* 分为两个阶段，分别是阶段 1 和阶段 2。阶段 1 初始化 DDR 并将 SBL 阶段 2 和器件管理器二进制文件加载到 DDR 中。SBL 阶段 2 有两个线程并行运行，一个 SBL 阶段 2 线程执行启动序列，启动硬件安全模块 (HSM) M4 内核、MCU-R5 内核以及在 A53 内核上运行的 Linux；另一个线程加载器件管理器，打开应用程序所需的驱动程序。

### 3.2 无闪烁切换

为了实现从 U-Boot 次级程序加载器 (SPL) 到 U-Boot 阶段的无闪烁切换，需保持图像帧缓冲区并且不要关闭显示子系统 (DSS) 驱动程序。为了将帧缓冲区从 SPL 阶段传递到 U-Boot 正常阶段，首先在内存中保留一块区域，并通过 video_post_probe 函数中的 bloblist 将该区域从 SPL 阶段传递到 U-Boot 正常阶段。在 SPL 阶段，帧缓冲区区域、大小、像素列数 (xsize) 以及像素行数 (ysize) 等各种参数都存储在一个 blob 中。Blob 是预留的内存区域，其中包含从一个阶段传递到另一个阶段的信息。当初始设置序列在 U-Boot 正常阶段运行时，会调用 reserve_video API，该 API 会检查视频 blob 是否存在。如果视频 blob 存在，应用程序编程接口 (API) 将使用来自前一个阶段的 blob 数据，确保设置相同的帧缓冲区区域和参数，从而保持启动画面不受任何闪烁影响，顺利过渡到下一个阶段。如果找到 blob，则不会再次探测 DSS 驱动程序，从而防止屏幕刷新。

## 4 从 SPL 到 U-Boot 的无闪烁切换

### 4.1 测试步骤

本节描述了实现早期启动屏幕以及从 SPL 阶段到 U-Boot 的无闪烁切换所需的步骤。从 Linux 9.0 软件开发者套件 (SDK) 开始，A53 SPL 的启动画面支持开箱即用。默认情况下，启动屏幕仅在 A53 SPL 时启用。默认的启动屏幕源设置为 SD 卡，并显示一个 gzip 压缩的 TI 标识 .bmp 图像。SPL 启动屏幕功能被编译到 tispl.bin 中，该文件在 U-Boot 编译过程中生成。对 SPL 启动屏幕功能所做的任何更改都需要重新编译 tispl.bin。使用新的 tispl.bin 启动电路板，以在 SPL 阶段查看启动屏幕。

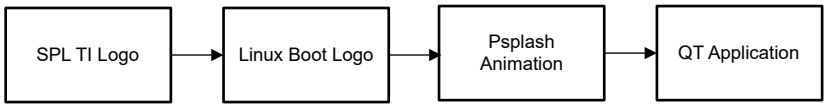


**图 4-1. 启动屏幕流**

与启动屏幕相关的所有信息都存储在 board.env 文件中，board/ti/am62px.env 包含 AM62P 器件的所有启动屏幕相关变量：

```
splashfile=ti_logo_414x97_32bpp.bmp.gz
splashimage=0x80200000
splashpos=m,m
splashsource=mmc
```

1. 显示自定义标识的说明：

   要显示自定义标识，请使用自定义标识的文件名更新启动屏幕文件。在 AM62P 的启动源为 MMC（即 SD 卡引导介质）的情况下，若需替换启动屏幕，请将新的启动屏幕镜像复制到 SD 卡的启动分区中。

   **备注**

   U-Boot 仅支持 .bmp 和压缩的 .bmp 图像。

2. 更改引导介质的说明：

   在 board/ti/am62px/evm.c 中定义的 splash_source struct 定义了可显示启动屏幕镜像的不同源。

```
static struct splash_location default_splash_locations[] = {
    {
        .name = "sf",
        .storage = SPLASH_STORAGE_SF,
        .flags = SPLASH_STORAGE_RAW,
        .offset = 0x700000,
    },
    {
        .name = "mmc",
        .storage = SPLASH_STORAGE_MMC,
        .flags = SPLASH_STORAGE_FS,
        .devpart = "1:1",
    },
};
```

要更改启动屏幕源，请使用 default_splash_locations struct 中定义的源的变量名更新 board.env 文件中的 splashsource 变量。AM62P 支持两种启动介质；“sf”指的是八通道串行外设接口 (OSPI)，而“mmc”指的是 SD 卡。使用从定义的源中选择的其中一个引导介质。要使用不同的引导介质，请在 struct 中添加信息，并使用新引导介质的名称更新 splashsource。

## 4.2 测量

从 POR 到显示画面出现的时间约为 680ms。默认情况下，通用输入输出 (GPIO) 引脚设置处于关闭状态。在执行 video_bmp_display() 函数（用于在面板上显示 .bmp 文件）后，该 GPIO 引脚的方向和数值被设为高电平。OSPI NOR 被用作测试时的引导介质。

- 用户扩展连接器上的 GPIO0_39（引脚 18）用于时间测量。要将该引脚的方向设置为输出，并将其状态设为高电平，可使用以下代码：

```
– a/common/bmp.c
+++ b/common/bmp.c
@@ -19,6 +19,7 @@
#include <splash.h>
#include <video.h>
#include <asm/byteorder.h>
+#include <asm/io.h>

/*

Allocate and decompress a BMP image using gunzip().
@@ -142,6 +143,11 @@ int bmp_display(ulong addr, int x, int y)
ret = video_bmp_display(dev, addr, x, y, align);
}

+ writel(0x00050007, 0x000F40A0);
+ writel(0xFFFFFF7F, 0x00600038);
+ writel(0x80, 0x00600040);
+
if (bmp_alloc_addr)
free(bmp_alloc_addr);
```

- 如果尚未启用 GPIO 驱动程序，可在 am62px_evm_r5_defconfig 文件中使用以下配置选项进行设置：

```
+CONFIG_SPL_GPIO=y
+CONFIG_GPIO=y
+CONFIG_DM_GPIO=y
+CONFIG_DA8XX_GPIO=y
+CONFIG_CMD_GPIO=y
```

- 要设置 GPIO 引脚编号 GPIO0_39 的引脚多路复用器 (pinmux)，请按照以下代码所述使用器件树引脚多路复用设置：

```
diff --git a/arch/arm64/boot/dts/ti/k3-am62p5-sk.dts b/arch/arm64/boot/dts/ti/k3-am62p5-sk.dts
index 4b8e7964ca4d..7dbf5e9b9c2b 100644
--- a/arch/arm64/boot/dts/ti/k3-am62p5-sk.dts
+++ b/arch/arm64/boot/dts/ti/k3-am62p5-sk.dts
@@ -232,6 +232,10 @@ hdmi_connector_in: endpoint {

 &main_gpio0 {
     bootph-all;
+
+    status = "okay";
+    pinctrl-names = "default";
+    pinctrl-0 = <&test_gpio_default>;
 };

 &main_gpio1 {
@@ -446,6 +450,12 @@ AM62PX_IOPAD(0x0078, PIN_OUTPUT, 1) /* (AC24) GPMC0_AD15.VOUT0_DATA23 */
             AM62PX_IOPAD(0x009c, PIN_OUTPUT, 1) /* (AD24) GPMC0_WAIT1.VOUT0_EXTPCLKIN */
         >;
     };
+
+    test_gpio_default: test-gpio {
+        pinctrl-single,pins = <
+            AM62PX_IOPAD(0x00a0, PIN_INPUT, 7) /* (P24) GPMC0_WPn.GPIO0_39 */
+        >;
+    };
 };

 &main_i2c0 {
@@ -789,6 +799,7 @@ &mcu_r5fss0_core0 {
 &main_uart0 {
     pinctrl-names = "default";
     pinctrl-0 = <&main_uart0_pins_default>;
+    test-gpios = <&main_gpio0 39 GPIO_ACTIVE_HIGH>;
     interrupts-extended = <&gic500 GIC_SPI 178 IRQ_TYPE_LEVEL_HIGH>,
             <&main_pmx0 0x1c8>; /* (D14) UART0_RXD PADCONFIG114 */
     interrupt-names = "irq", "wakeup";
```

- 接下来，将 GPIO 引脚和 MCU_PORz 连接至逻辑分析仪，并测量两者之间的时间差，以获取精确的时间戳。

# 5 从 SBL 到 Linux® 内核的无闪烁转换

## 5.1 测试步骤

本节描述了实现早期启动屏幕以及从 SBL 阶段到 Linux 内核的无闪烁切换所需的步骤。MCU+ SDK 中的 DSS 共享示例将镜像的早期启动屏幕与 SBL 集成在一起，并支持 OSPI 引导介质、器件管理器和处理器间通信功能。引导加载程序、IPC 和显示功能会在独立的任务中运行。显示任务通过 Alpha 混合显示启动画面，并最终切换到显示共享任务，在该任务中，信号帧会快速来回移动。该示例使用了 Falcon 引导，这意味着跳过了中间的 U-Boot 阶段，SBL 直接启动 Linux 镜像。本 DSS 示例经过一些修改后用于演示。

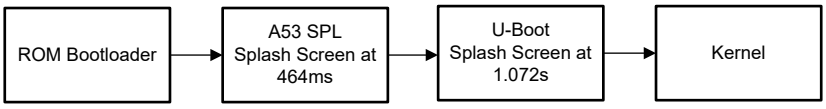


**图 5-1. 启动界面时间**

- 下载处理器 SDK 并使用 /board-support 下的 ti-linux-kernel 目录按照以下步骤进行修改。在对 ti-linux-kernel 进行修改后，生成设备树 Blob (DTB) 和镜像文件。这些文件稍后用于创建 linux.appimage，该文件在 RTOS 示例中用于在 A53 内核上运行 Linux。
- linux.appimage 是使用 Falcon 引导模式构建的。因此，将 bootargs 信息包含在所选节点下的 k3-am62p5-sk.dts 文件中：

```
bootargs = "console=ttyS2,115200n8 earlycon=ns16550a,mmio32,0x02800000 root=/dev/mmcblk1p2 rw
rootfstype=ext4 rootwait";
```

- 为了确保在 Linux 内核启动期间启动画面保持不变，ti-u-boot 使用帧缓冲区区域元数据动态更新 Linux 内核器件树，并将该区域标记为在 Linux 器件树中保留，如下所示：

```
framebuffer: framebuffer@ff700000 {
     reg = <0x00 0xff700000 0x00 0x008ca000>;
     no-map;
};
```

- 通过手动修改所选节点下的板级器件树文件，将 simple-framebuffer 节点的状态设置为 *“oke”*，如以下代码所示：

```
framebuffer0: framebuffer@0 {
      compatible = "simple-framebuffer";
      power-domains = <&k3_pds 186 TI_SCI_PD_EXCLUSIVE>,
                      <&k3_pds 243 TI_SCI_PD_EXCLUSIVE>,
                      <&k3_pds 244 TI_SCI_PD_EXCLUSIVE>;
      clocks = <&k3_clks 186 6>,
               <&dss0_vp1_clk>,
               <&k3_clks 186 2>;
      display = <&dss0>;
      reg = <0x00 0xff700000 0x00 0x008ca000>;
      width = <1920>;
      height = <1200>;
      stride = <(1920 * 4)>;
      format = "x8r8g8b8";
 };
```

- 为了在显示服务器启动之前保持启动动画的显示，需要通过在 arch/arm64/configs/defconfig 中删除以下配置选项来手动禁用 *直接渲染管理器 (DRM) 帧缓冲器设备仿真* 功能，如下所述：# CONFIG_DRM_FBDEV_EMULATION 未设置
- 在之前展示的 Linux 目录更改之后，构建 Linux 内核，以创建 DTB 和内核镜像。
- 将覆盖文件 k3-am62p5-sk-microtips-mf101hie-panel.dtbo 应用到 DTB 文件，以支持在 OLDI 面板上的显示。使用以下命令：

```
fdtoverlay -i ./arch/arm64/boot/dts/ti/k3-am62p5-sk.dtb ./arch/arm64/boot/dts/ti/k3-am62p5-sk-
microtips-mf101hie-panel.dtbo -o ./../../board-support/prebuilt-images/am62pxx-evm-display-
cluster/k3-am62p.dtb
```

- 将以下两个文件复制到 /board-support/prebuilt-images/am62pxx-evm-display-cluster 文件夹：
  1. k3-am62p.dtb ( 在上一步中创建 )
  2. 镜像 (arch/arm64/boot//)
- 在 examples/drivers/dss/dss_display_share/dss_display_share.c 文件中，删除 DispApp_splashThread() 和 DispApp_displayShareThread() 的使用和定义

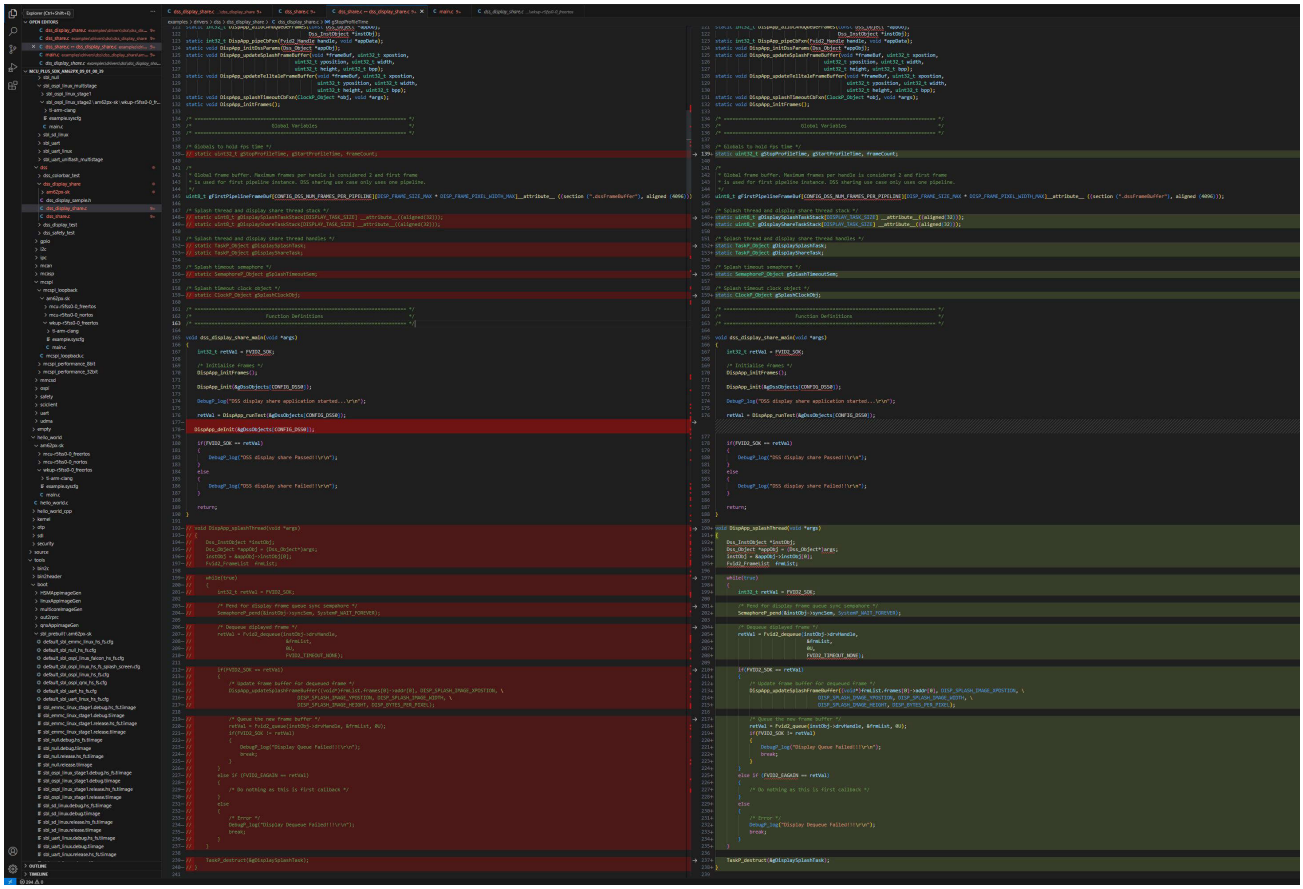

图 5-2. 代码更改 1

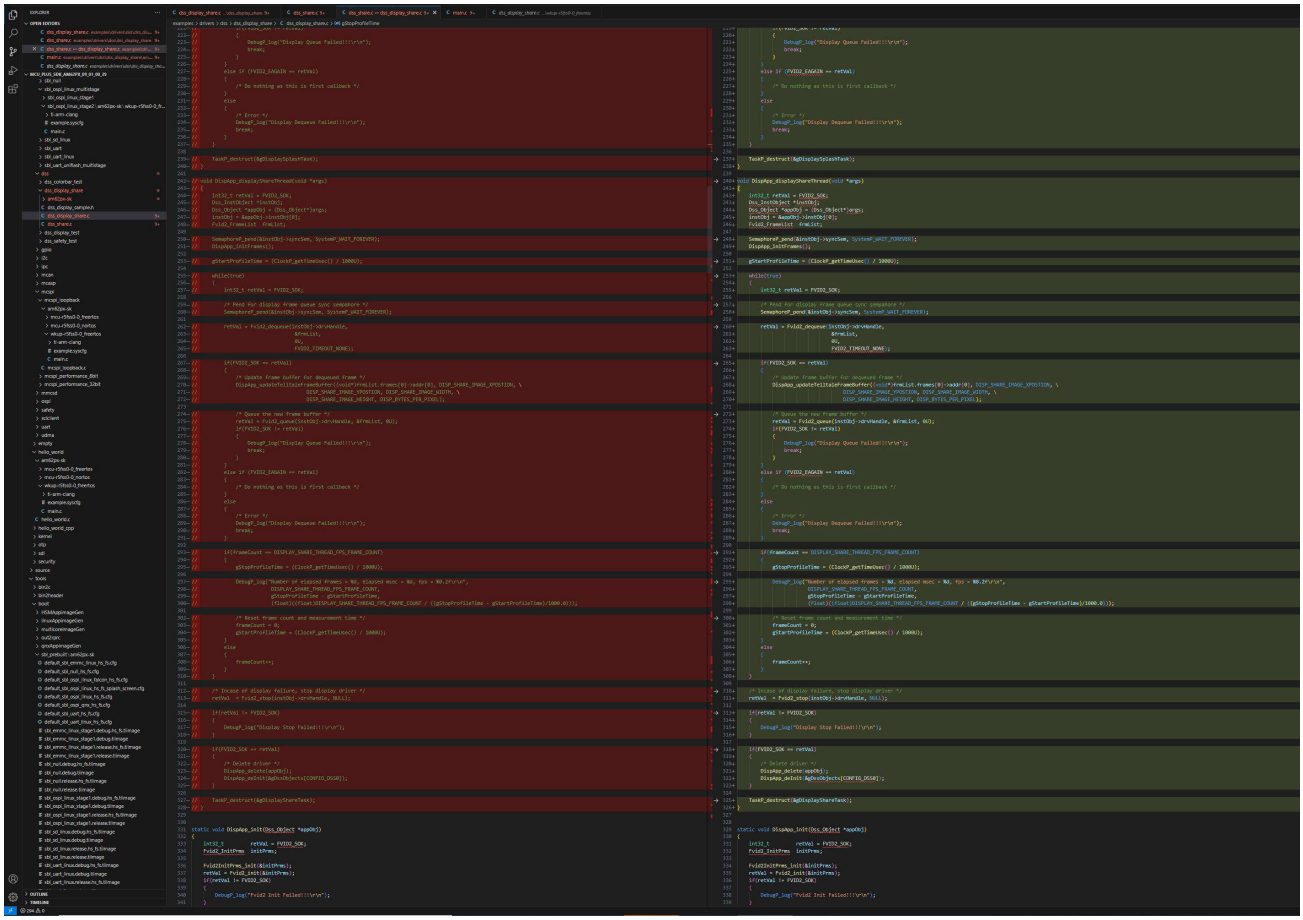

图 5-3. 代码更改 2

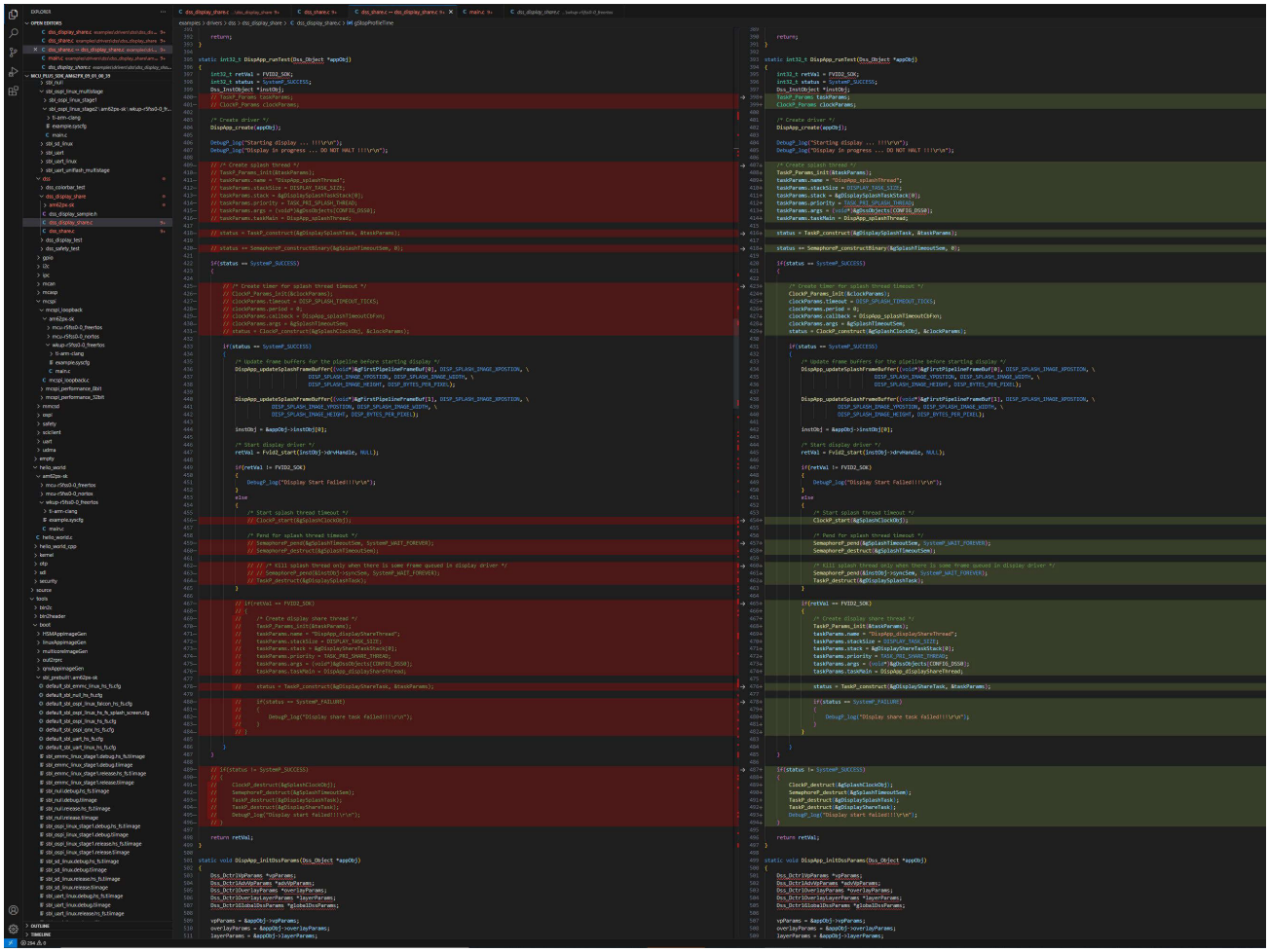

**图 5-4. 代码更改 3**

- 在 tools/boot/sbl_prebuild/am62px-sk/default_sbl_ospi_linux_hs_fs_splash_screen.cfg 中编译镜像
- 使用通用异步收发器 (UART) TI UniFlash 工具将构建好的镜像烧录到 OSPI 闪存中。
- 为了快速渲染 Linux GUI 应用程序，使用 SDK 中的顶层 makefile 构建并安装 ti-img-rogue-driver。采用以下步骤：
  1. 运行命令：
     make ti-img-rogue-driver (Value RGX_BVNC="36.53.104.796" in Rules.make)
  2. 插入 SD 卡并运行以下命令：
     sudo make ti-img-rogue-driver_install DESTDIR=/media/aparna/root
- 切换到 OSPI NOR 引导模式以观看演示。
  BOOTMODE [ 8 : 15 ] (SW5) = 0000 0000
  BOOTMODE [ 0 : 7 ] (SW4) = 1100 1110

## 5.2 测量

从 POR 到显示画面出现的时间约为 180ms。该测量是通过使用一个默认处于低电平状态的 GPIO 引脚进行的。该引脚通过 CSL_dssVpSetGoBit() 函数使用 gpio_set_high() API 设置为高电平。OSPI NOR 被用作测试时的引导介质。

- 本实验使用默认 MCU+ SDK 中提供的 dss_display 共享示例。对该示例进行了修改，以实现通过 SBL 流程演示的无闪烁切换。
- 用户扩展连接器上的 GPIO0_39（引脚 18）用于时间测量。图 5-5 展示了 SysConfig 中用于启用此引脚的选项。

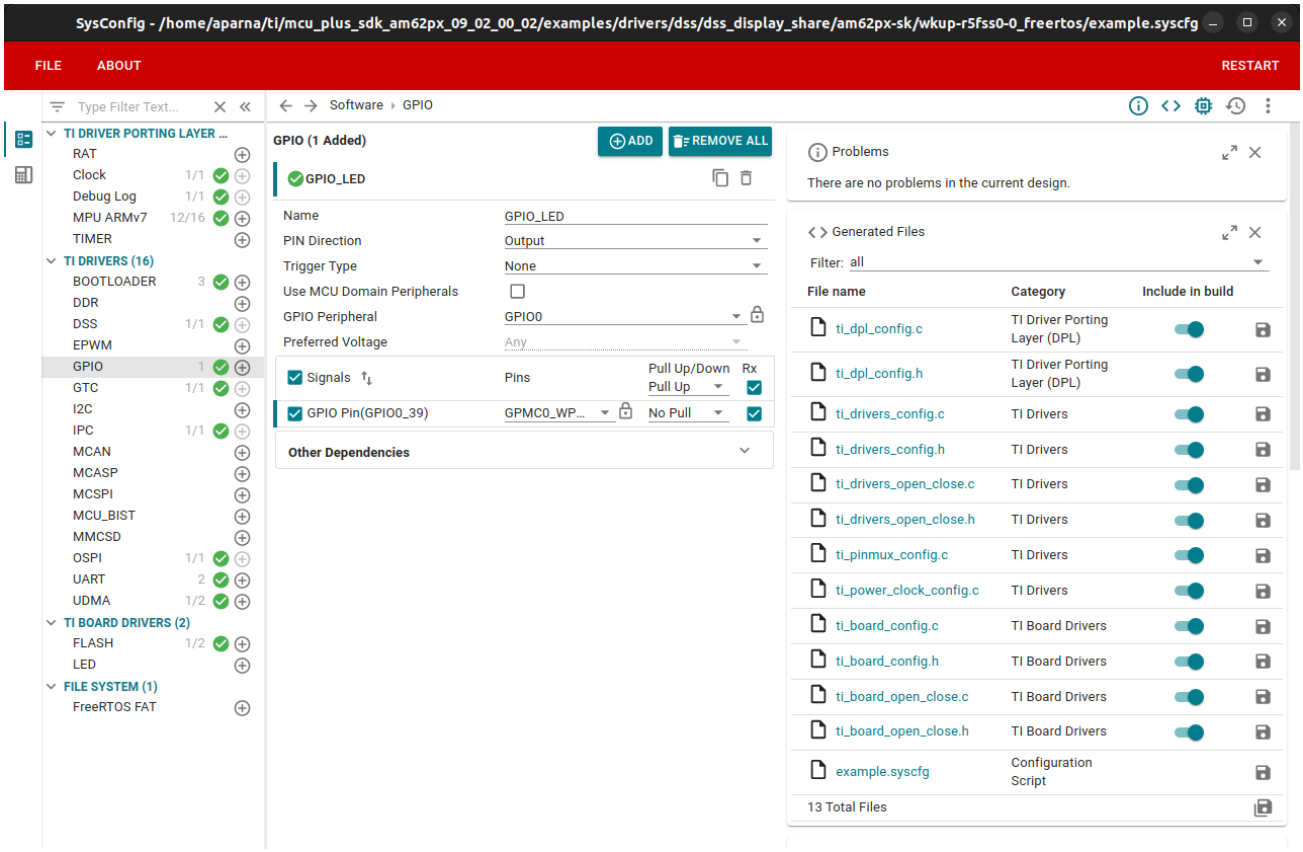


**图 5-5. SysConfig 设置**

- 通过在 dss_display_share.c 中定义 API，将 GPIO 引脚设置为高电平，如下所示：

```
void gpio_set_high(void *args) {
    uint32_t gpioBaseAddr, pinNum;
    DebugP_log("GPIO LED started ...\r\n");

    /* Get address after translation translate */
    gpioBaseAddr = (uint32_t) AddrTranslateP_getLocalAddr(GPIO_LED_BASE_ADDR); pinNum =
GPIO_LED_PIN;
    GPIO_setDirMode(gpioBaseAddr, pinNum, GPIO_LED_DIR);
    GPIO_pinWriteHigh(gpioBaseAddr, pinNum);
    DebugP_log("GPIO LED HIGH!!\r\n");
}
```

- 使用 CSL_dssVpSetGoBit() 中的 API，如下所示：

```
diff --git a/source/drivers/dss/v0/hw_include/V3/csl_dssVideoPort.c b/source/drivers/dss/v0/
hw_include/V3/csl_dssVideoPort.c

index f882d54..ee18b95 100755

--- a/source/drivers/dss/v0/hw_include/V3/csl_dssVideoPort.c

+++ b/source/drivers/dss/v0/hw_include/V3/csl_dssVideoPort.c

@@ -183,6 +183,8 @@ void CSL_dssVpEnable(CSL_dss_vpRegs *vpRegs, uint32_t enable)

     CSL_REG32_WR(&vpRegs->CONTROL, regVal);

 }

+extern void gpio_set_high(void *args);

+

 void CSL_dssVpSetGoBit(CSL_dss_vpRegs *vpRegs)

 {

     uint32_t regVal;

@@ -192,6 +194,7 @@ void CSL_dssVpSetGoBit(CSL_dss_vpRegs *vpRegs)

               DSS_VP1_CONTROL_GOBIT,

               CSL_DSS_VP1_CONTROL_GOBIT_VAL_UFPSR);

     CSL_REG32_WR(&vpRegs->CONTROL, regVal);

+    gpio_set_high(NULL);

 }

 void CSL_dssVpSetLcdTdmConfig(CSL_dss_vpRegs *vpRegs,
```

- 当 GPIO0_39 引脚连接到逻辑分析仪时，GPIO 引脚大约在 180ms 时被设置为高电平。

## 6 结果

总之，通过实现无闪烁过渡的早期启动屏幕，可以提供流畅且视觉上吸引人的应用程序入口点，从而提升用户体验。通过优化加载顺序并确保流畅过渡，开发人员可以降低用户感知的等待时间，从而保持用户的参与度。未来的开发应重点优化这些过渡效果，并探索更多优化技术，以提升性能和响应速度。

**表 6-1. 启动界面显示时间对比**

| 引导流程中的启动界面显示情况 | 所需时间 |
|---|---|
| SBL 流程中的 SBL 启动界面显示情况 | 180ms |
| SPL 流程中的 U-Boot 启动界面显示情况 | 680ms |


**图 6-1. 硬件设置**

# 重要通知和免责声明

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 产品的相关应用。 严禁以其他方式对这些资源进行复制或展示。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