



Aparna Patra, Devarsh Thakkar, Nikhil Jain, Soumya Tripathy

Sitara MPU

ABSTRACT

This application note provides information on how to display a splash screen as fast as possible on AM62P and perform a glitch-free transition from the splash screen to the full system user interface (UI). The steps in this document are demonstrated using the Universal Boot Loader (U-Boot) flow as well as custom Slim Bootloader (SBL) along with the Linux® Kernel.

Table of Contents

1 Introduction	2
2 Hardware Used	2
2.1 AM62Px Processor.....	2
2.2 SK-LCD1.....	3
2.3 Display Subsystem on AM62P.....	3
3 Early Splash-Screen Architecture	3
3.1 Boot Stages on AM62P.....	3
3.2 Flicker-Free Transition.....	4
4 Flicker-Free Transition From SPL to U-Boot	4
4.1 Steps to Test.....	4
4.2 Measurements.....	5
5 Flicker-Free Transition From SBL to Linux® Kernel	6
5.1 Steps to Test.....	6
5.2 Measurements.....	11
6 Results	12

List of Figures

Figure 2-1. AM62P Block Diagram.....	2
Figure 2-2. DSS Block Diagram.....	3
Figure 3-1. U-Boot.....	3
Figure 4-1. Splash Flow.....	4
Figure 5-1. Splash Timings.....	6
Figure 5-2. Code Changes 1.....	8
Figure 5-3. Code Changes 2.....	9
Figure 5-4. Code Changes 3.....	10
Figure 5-5. SysConfig Settings.....	11
Figure 6-1. Hardware Setup.....	13

List of Tables

Table 6-1. Splash Display Time Comparison.....	12
--	----

Trademarks

Sitara™ is a trademark of Texas Instruments.

Linux® is a registered trademark of Linus Torvalds.

FreeRTOS® is a registered trademark of Amazon Technologies, Inc.

Arm® and Cortex® are registered trademarks of Arm Limited.

All trademarks are the property of their respective owners.

1 Introduction

Increasing numbers of automotive, industrial, and robotic use-cases require enabling the display as early as possible in the boot cycle. The operating systems involve multiple stages for boot up and a bootloader is the first software component to come up and initialize the system before booting up to the full system. This application report explains how to enable the display at the bootloader stage, display a splash-screen or animation at this stage, and have a flicker-free transition to the system UI. The splash screen enablement is explained for U-Boot which is a widely used open-source bootloader, as well as a Secondary Bootloader (FreeRTOS®-based) which can perform a fast boot up to Linux using the Falcon mode boot flow. This guide then explains how the splash screen display context can be preserved while the system is booting up and a flicker-free transition to the system UI can be achieved.

2 Hardware Used

2.1 AM62Px Processor

The AM62Px (P = Plus) is an extension of the existing Sitara™ AM62x low-cost family of application processors built for high-performance embedded 3D display applications. Scalable Arm® Cortex®-A53 performance and embedded features, such as: multiscreen high-definition display support, 3D-graphics acceleration, 4K video acceleration, and extensive peripherals make the AM62Px well-designed for a broad range of automotive and industrial applications, including automotive digital instrumentation, automotive displays, industrial Human Machine Interfaces (HMI), and more.

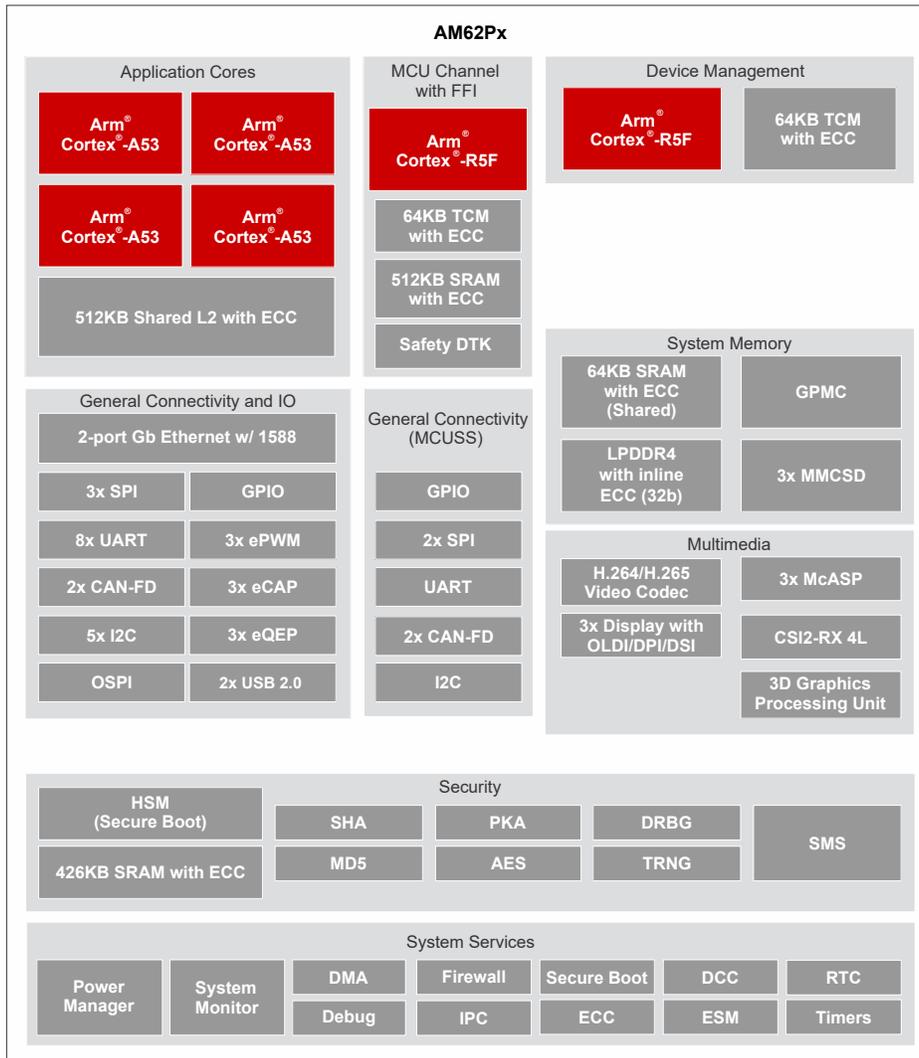


Figure 2-1. AM62P Block Diagram

2.2 SK-LCD1

The 1920 × 1200 Open LVDS Display Interface (OLDI) display or Low Voltage Differential Signaling (LVDS) Liquid Crystal Display (LCD) kit is an add-on accessory for the starter kit AM62x processor Evaluation Modules (EVM) to add touch and display functions for the evaluation of HMI, industrial PC, and other use cases requiring display. This model is composed of a Thin Film Transistor (TFT) LCD panel, a driving circuit, a backlight system and a projected capacitive touch panel.

2.3 Display Subsystem on AM62P

The Display Subsystem (DSS) is a flexible, multipipeline subsystem that supports high-resolution display outputs. DSS includes input pipelines providing multilayerblending with transparency to enable on-the-fly composition. Various pixel processing capabilities are supported, such as color space conversion and scaling, among others. DSS includes a Direct Memory Access (DMA) engine, which allows direct access to the framebuffer (device system memory). Display outputs can connect seamlessly to an Open LVDS Display Interface transmitter (OLDITX), or can directly drive device pads as a Display Parallel Interface (DPI).

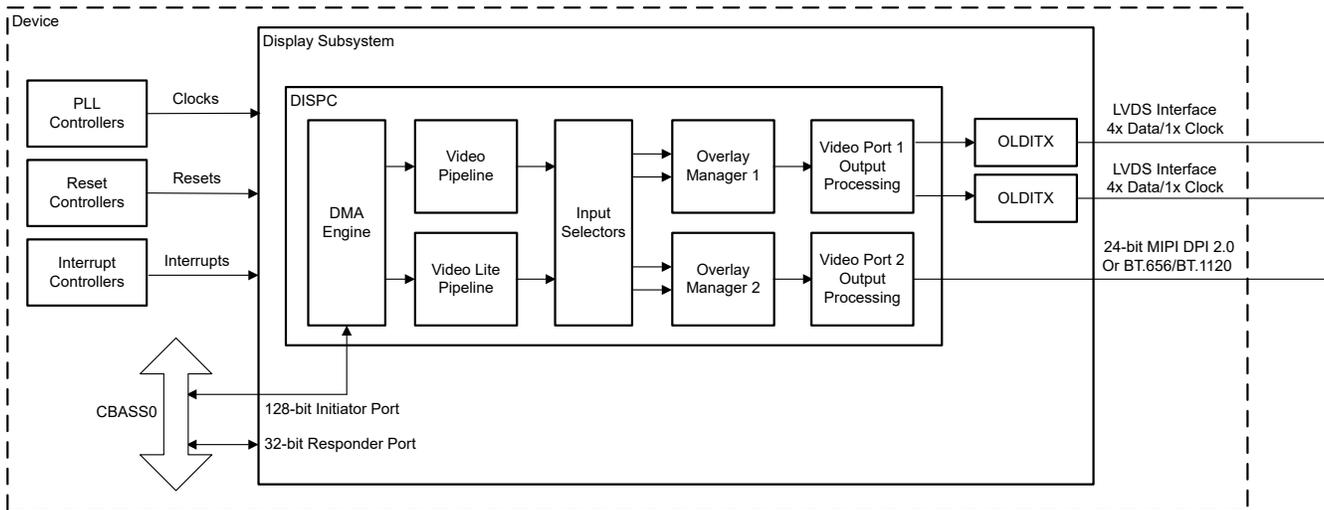


Figure 2-2. DSS Block Diagram

3 Early Splash-Screen Architecture

3.1 Boot Stages on AM62P

This section describes the two boot flow sequences used on AM62P. The Read Only Memory (ROM) code is the first block of code that is automatically run on device start-up or after power-on reset (POR). The ROM bootloader code is hardcoded into the device. This is a very small binary, due to the limited amount of internal memory. After the ROM bootloader comes up, the bootloader either loads the *Secondary Program* loader for a Linux-specific boot flow, or the *Secondary Bootloader* for Real-Time Operating System (RTOS) based boot flow. These boot sequences can also be customized according to the required use case.

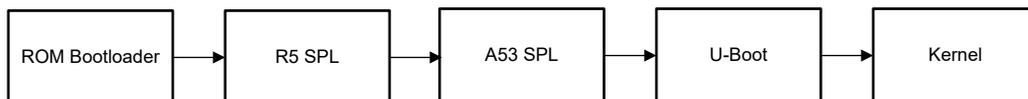


Figure 3-1. U-Boot

1. *Secondary Program* loader mainly serves to initialize the external Double Data Rate (DDR) memory and sets up the boot process for the next bootloader stage, U-Boot. U-Boot, running out of DDR, provides wider functionalities, such as command (CMD) line support, device driver infrastructure and Kconfig infrastructure. U-Boot then loads the Kernel image, to start a High-Level Operating System (HLOS) like Linux on a A53 core.

2. *Secondary Bootloader* is a FreeRTOS based bootloader, responsible for performing device-specific initialization, loading of respective binaries to initialize the subsequent cores, and to ultimately start the application. On AM62P, the *Secondary Bootloader* is divided into two stages, namely stage1 and stage 2. Stage 1 initializes DDR and loads the SBL stage 2 and the device manager binary to DDR. The SBL stage 2 has two threads running in parallel, one SBL stage 2 thread to run the boot sequence to start the Hardware Security Module (HSM) M4 core, MCU-R5 core, and Linux on the A53 core, another thread loads the device manager that opens the drivers required for application.

3.2 Flicker-Free Transition

To achieve flicker-free transition from U-Boot Secondary Program Loader (SPL) to the U-Boot stage, preserve the image framebuffer and do not close the Display Subsystem (DSS) driver. To pass the framebuffer from the SPL stage to the U-Boot proper stage, a region in the memory is reserved and the same region is passed from the SPL stage to U-Boot proper using the `bloblist`, in the `video_post_probe` function. At the SPL stage, various parameters like framebuffer region, size, number of pixel columns (`xsize`), and the number of pixel rows (`ysize`) are all stored in a blob. The blobs are reserved memory regions which contain information to be passed from one stage to another. When the initial setup sequence is running in the U-Boot proper stage, `reserve_video` API is called, which determines if video blob is present. If video blob is present, the Application Programming Interface (API) uses blob data from the previous stage which makes sure the same framebuffer region and parameters are set, thus keeping the splash image intact without any flicker across the stage. If blob is found, the DSS driver is not probed again, which prevents screen refresh.

4 Flicker-Free Transition From SPL to U-Boot

4.1 Steps to Test

This section describes the steps required to achieve an early splash screen, along with flicker-free transition from the SPL stage to U-Boot. The splash screen support from A53 SPL is provided out-of-the-box in the Linux version 9.0 Software Development Kit (SDK) onwards. By default, the splash screen is only enabled at A53 SPL. The default splash source was set to an SD card and displays a gzip TI logo .bmp image. The SPL splash screen features are compiled in the `tispl.bin` which is built during U-Boot compilation. Any changes made to the SPL splash screen feature requires recompiling `tispl.bin`. Use the new `tispl.bin` to boot the board to see the splash screen at the SPL stage.

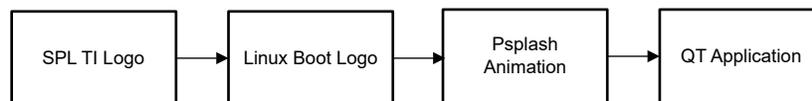


Figure 4-1. Splash Flow

All information related to splash screens is stored in a `board.env` file, `board/ti/am62px.env` contains all the splash-related variables for the AM62P device:

```

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

1. Instructions for displaying a custom logo:

To display a custom logo update the splash file with the file name of the custom logo. In a case where the AM62P splash source is MMC (this refers to SD card boot media) to replace the image copy the new splash image in the boot partition of the SD card.

Note

U-Boot only supports .bmp and compressed .bmp images.

2. Instructions for changing boot media:

The `splash_source` struct defined in `board/ti/am62px/evm.c` defines the different sources from which the splash image can be displayed.

```
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",
    },
};
```

To change the splash source, update the `splashsource` variable in `board.env` file with the variable name of the sources defined in the `default_splash_locations` struct. AM62P supports two boot medias; "sf" refers to Octal Serial Peripheral Interface (OSPI) and "mmc" refers to an SD card. Use one of the boot media from the sources defined. To use a different boot media, add the information in the struct and update the `splashsource` with the name of the new boot media.

4.2 Measurements

The time taken from POR to when the display comes up is noted to be approximately 680ms. The General Purpose Input-Output (GPIO) pin settings, by default, are in Off state. The direction and value of this GPIO pin is set to high, just after the `video_bmp_display()` function, which displays a .bmp file onto the panel. OSPI NOR is used as boot-media for testing purposes.

- GPIO_39 on the user expansion connector-pin 18, is used for time measurements. To set the direction of this pin to output, and the state of this pin to high, use the following code:

```
- 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(0xFFFFF7F, 0x00600038);
+ writel(0x80, 0x00600040);
+
if (bmp_alloc_addr)
free(bmp_alloc_addr);
```

- To enable the GPIO driver, if not done already, use the following config options in the `am62px_evm_r5_defconfig` file:

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

- To set the pin multiplexer (pinmux) of the GPIO pin number GPIO0_39, use the device tree pinmux settings as mentioned in the following code:

```
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_pmux0 0x1c8>; /* (D14) UART0_RXD PADCONFIG114 */
        interrupt-names = "irq", "wakeupt";
```

- Next, connect the GPIO pin and MCU_PORZ to the logic analyzer and measure the time difference between the two to get the accurate timestamp.

5 Flicker-Free Transition From SBL to Linux® Kernel

5.1 Steps to Test

This section describes the steps required to achieve an early splash screen, along with flicker-free transition from the SBL stage to the Linux Kernel. The DSS sharing example in the MCU+ SDK integrates early splash of image along with SBL on the OSPI boot media, Device Manager, and Inter-processor communication functionality. The bootloader, IPC, and Display run on separate tasks. The Display task displays a splash image with alpha blending and finally switches to the display sharing task, where telltale frames quickly move back and forth. Falcon boot is used in the example, which means the intermediate U-Boot stage is skipped and SBL directly boots the Linux image. This DSS example, with a few modifications is used for demonstration.

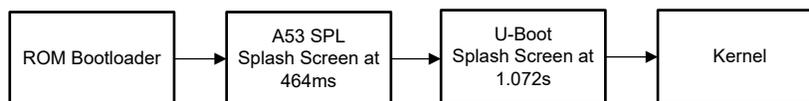


Figure 5-1. Splash Timings

- Download the processor SDK and use the `ti-linux-kernel` directory under `/board-support` to make the modifications as described in the following steps. Generate the Device Tree Blob (DTB) and image file after the modifications to `ti-linux-kernel`. These files are later used to create the `linux.appimage`, which is used in the RTOS example to run Linux on an A53 core.
- The `linux.appimage` is built using Falcon boot mode. Hence, include the bootargs information in the `k3-am62p5-sk.dts` file under the chosen node:

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

- To make sure that the splash screen remains persistent while the Linux Kernel boots up, `ti-u-boot` dynamically updates the Linux Kernel device-tree with framebuffer region meta-data, marking the region as reserved in the Linux device tree as follows:

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

- Set the status of the `simple-framebuffer` node to "okay" by manually modifying the board device-tree file under the chosen node as shown in the following code:

```
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";
};
```

- To keep the boot animation alive until the display server starts up, the *Direct Rendering Manager (DRM) framebuffer device emulation* feature needs to be manually disabled by removing the following config option in `arch/arm64/configs/defconfig` like mentioned below: `# CONFIG_DRM_FBDEV_EMULATION` is not set
- After the previously shown changes in Linux directory, build the Linux Kernel, to create the DTB and Kernel image.
- Apply the overlay file `k3-am62p5-sk-microtips-mf101hie-panel.dtbo` onto the DTB file to support display on an OLDI panel. Use the following command:

```
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
```

- Copy the following two files into the `/board-support/prebuilt-images/am62pxx-evm-display-cluster` folder:
 - `k3-am62p.dtb` (created in the previous step)
 - Image (`arch/arm64/boot/`)
- Remove the usage and definition of `DispApp_splashThread()` and `DispApp_displayShareThread()` in the `examples/drivers/dss/dss_display_share/dss_display_share.c` file



```

// ... (old) ...
void display_start(void)
{
    // ... (old) ...
}

// ... (old) ...
void display_stop(void)
{
    // ... (old) ...
}

// ... (old) ...
void display_refresh(void)
{
    // ... (old) ...
}

// ... (new) ...
void display_start(void)
{
    // ... (new) ...
}

// ... (new) ...
void display_stop(void)
{
    // ... (new) ...
}

// ... (new) ...
void display_refresh(void)
{
    // ... (new) ...
}
  
```

Figure 5-3. Code Changes 2

5.2 Measurements

The time taken from POR to when the display comes up is noted to be approximately 180ms. The measurement was taken by using a GPIO pin, which by default is in active-low state. The pin is set high in `CSL_dssvpSetGoBit()` by using the `gpio_set_high()` API. OSPI NOR is used as boot-media for testing purposes.

- The `dss_display` share example provided in the default MCU+ SDK is used for experimentation. Modifications to this example are made to implement the flicker-free transition demonstrated through SBL flow.
- GPIO0_39 on the user expansion connector-pin 18, is used for time measurements. [Figure 5-5](#) shows the options in SysConfig to enable this pin:

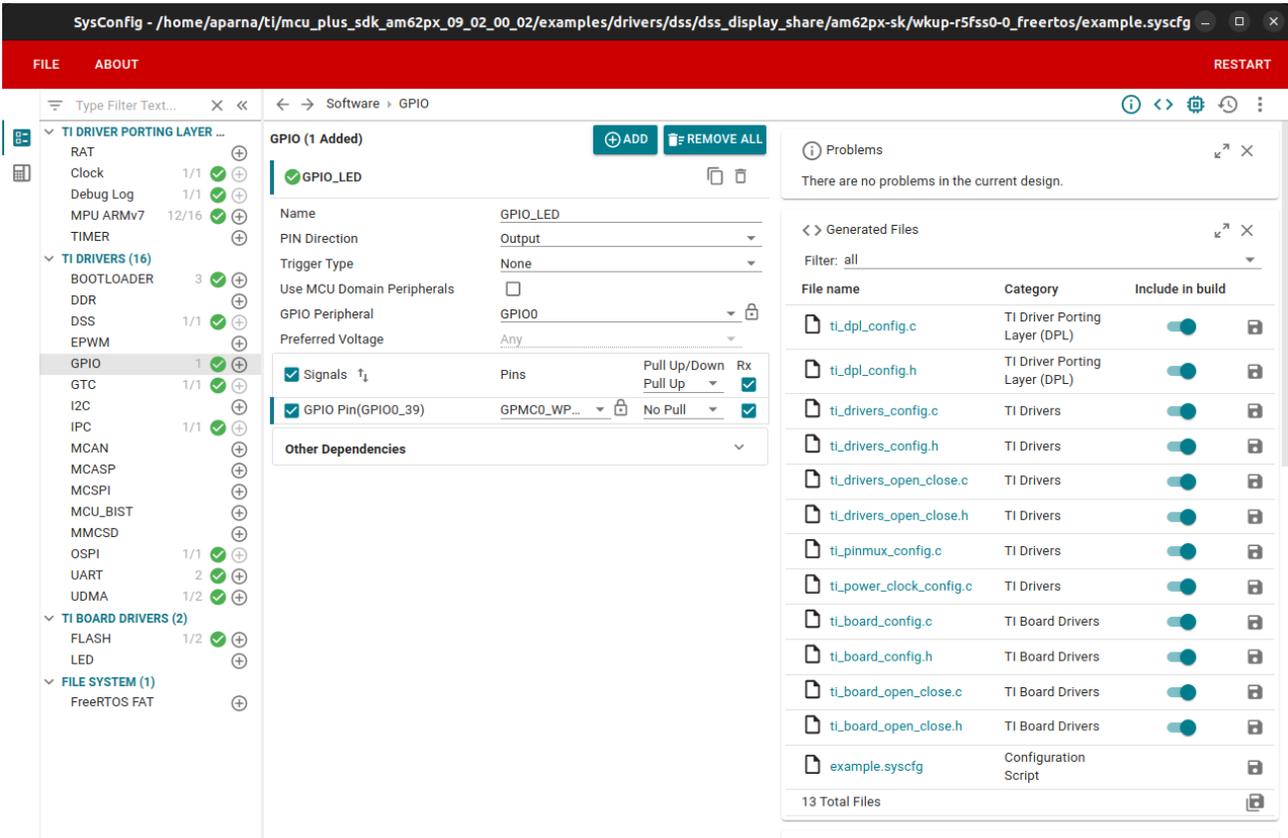


Figure 5-5. SysConfig Settings

- Set the GPIO pin to high by defining the API in `dss_display_share.c` as follows :

```

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");
}
  
```

- Use the API in `CSL_dssVpSetGoBit()` as follows:

```
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,
```

- When the GPIO0_39 pin is connected to the logic analyzer, the GPIO pin is set high at around 180ms.

6 Results

In conclusion, implementing an early splash screen with a flicker-free transition enhances the user experience by providing a seamless and visually appealing entry point into the application. By optimizing the loading sequence and providing smooth transitions, developers can reduce perceived wait times and maintain user engagement. Focus future developments on refining these transitions and exploring further optimization techniques to enhance performance and responsiveness.

Table 6-1. Splash Display Time Comparison

Splash Displayed From Boot Flow	Time Taken
Splash from SBL in SBL flow	180ms
Splash from U-Boot in SPL flow	680ms



Figure 6-1. Hardware Setup

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), 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 will 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](#) or other applicable terms available either on [ti.com](https://www.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.

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

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2025, Texas Instruments Incorporated