Application Note

在 AM261x 上实现基于 VBUS 的 USB 主机检测

Shaunak Deshpande, Tejas Kulakarni

摘要

AM261x Sitara™ Arm® 微控制器属于 Sitara AM26x 实时 MCU 系列，旨在满足下一代工业和汽车嵌入式系统的严苛处理需求。AM261x 基于可扩展的 Arm® Cortex®-R5F 内核，通过广泛的外设、集成安全特性和灵活的通信接口，提供高性能实时控制。其中，USB（通用串行总线）提供了广泛采用的高速串行接口来连接器件。

USB 通过标准化 USB 主机和 USB 器件之间的电力输送和数据传输，为连接各种消费类、工业和汽车器件提供了设计。正确的主机检测对于确保 USB 器件仅在存在有效主机时进行初始化至关重要，这能防止合规性问题并提高系统稳健性。

在 AM261x 上，USB 控制器支持使用外部 VBUS 电压运行。但是，默认实现无法主动监测 VBUS 电压，以进行主机检测。相反，无论提供 VBUS 电压的主机是否存在，USB 驱动程序代码都无条件地将控制器寄存器设置为 PHY 上电，并启用 USB 模块。虽然这种方法简化了初始化，但也存在一些缺点，例如虚假枚举尝试、总线争用、功耗增加、USB 规范不合规等。

本应用手册演示了一个参考实现方案，该方案通过一个简单的 GPIO 实现，修改了 AM261x 默认的 USB 行为，加入了通过 VBUS 监测实现的真正主机检测功能。采用此实现方式，该器件仅在确认存在有效主机和 VBUS 电压后，启用 PHY 并启动枚举。

本应用手册中描述的设计作为参考设计提供。尚未对 AM261x 进行广泛的 USB 应力测试。我们鼓励客户根据终端应用的情况调整、验证和鉴定该方法。

内容

商标

1 简介

有关 AM261x 器件上 USB 子系统的深入说明，请参阅 AM26x MCU+ Academy USB 模块、技术参考手册 (TRM) 和 MCU+ SDK 文档。图 1-1 显示了 AM261x 上 USB 2.0 子系统 (USB2SS) 的功能方框图，其中包括包装器模块、USB 控制器模块、PHY 模块、外部接口和内部接口。

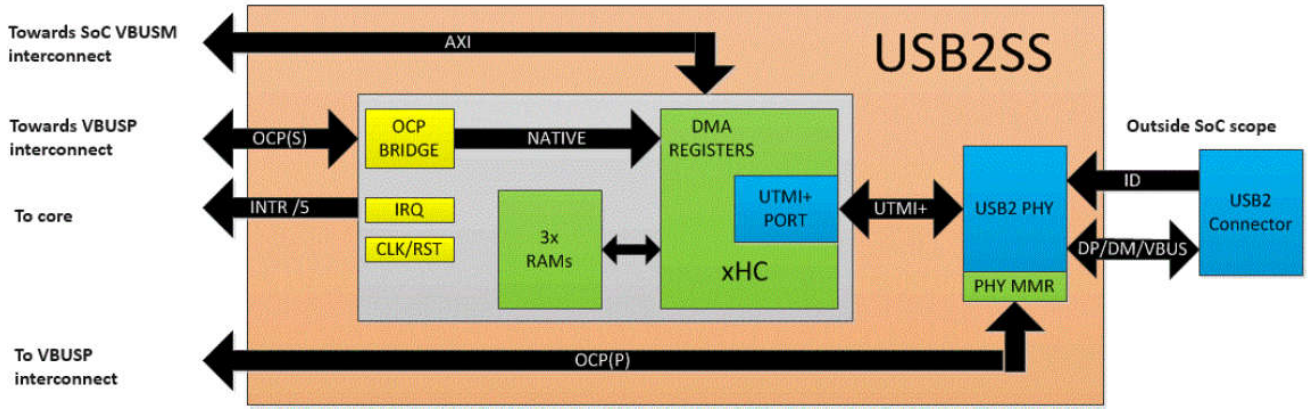

图 1-1. AM261x USB 子系统

AM261x USB2SS 依靠外部电源逻辑，通过外部电源生成 5V VBUS。该器件提供专用输出信号 USB0_DRVVBUS，为有效高电平，用于启用或禁用外部 VBUS 电源。

在主机模式下运行时，USB 控制器会自动将 USB0_DRVVBUS 驱动为高电平，从而启用外部电荷泵，并在 VBUS 线路上提供 5V 电压。

在器件模式下运行时，控制器会将 USB0_DRVVBUS 驱动为低电平、禁用外部电源并确保 AM261x 不会为 VBUS 供电。

此控制完全由 USB 控制器处理，并且在硬件连接和软件初始化配置正确的情况下，对用户透明。

请注意，USB 控制器为自供电（通过 AM261x 器件电源引脚）。在器件模式下，控制器不依赖于外部提供的 VBUS 电压来保持运行。相反，它通过寄存器在内部传达 VBUS 的存在：MSS_CTRL.CONTROL_USBOTGHS_CONTROL.CONTROL_USBOTGHS_CONTROL_VBUSVALID (0x50D00894)

该寄存器位用于将 VBUSVALID 信号驱动到 USB 控制器中。由于 AM261x 在器件模式下不暴露物理 VBUS 输入引脚，因此必须在软件中配置该位。默认情况下，AM261x USB 驱动程序会在 USB_init() 期间设置 VBUSVALID 位，而不验证 USB 主机的实际存在。虽然这会启用控制器和 PHY，但会绕过 USB 规范定义的正常主机检测机制。

这种默认行为会带来一些风险：

- 虚假枚举尝试：即使未连接主机，器件也可能启动枚举。
- 总线争用：该器件可以在没有有效主机提供 VBUS 的情况下驱动 D+/D– 线，这违反了 USB 协议规则
- 功耗增加：PHY 仍然不必要地通电，从而影响低功耗和电池敏感型应用
- 不符合 USB 规范：USB 标准明确要求器件在检测到 VBUS 之前保持不活动状态。

为了验证运行是否可靠且符合规范，该器件必须根据监测 VBUS 信号，实现真正的主机检测。在正确检测机制下，器件仅在检测到有效主机时才启用 PHY 并开始枚举，从而避免前述问题。

2 USB 主机检测

2.1 通用 USB 主机检测流程

USB（通用串行总线）定义了强大的机制，用于检测何时连接主机和器件，并在进行任何枚举之前建立所需的电气条件。在传统实现中，这种检测主要通过监测 VBUS 线路和 D+/D– 差分对来实现，确保只有在确认有效的主机器件关系后，控制器才会启用数据通信。VBUS 线路是主机提供的主电源轨。在标准 USB 2.0 实现中，主机以 USB 规范定义的电流能力，在 VBUS 上提供 5V 电压（对于高速 USB 2.0，通常为 500mA，对于 USB 3.x 则更高）。器件不应驱动这条线路。

传统 USB 器件控制器包含 VBUS 检测电路，该电路可检测是否存在 5V 信号。可使用以下两种方法之一执行检测：

- 基于比较器的检测：内部比较器持续监测 VBUS。一旦电压上升到超过特定阈值（例如，规范定义的最小值为 4.4V），控制器就会设置状态标志，指示存在有效的主机连接。
- 专用 VBUSVALID 输入：许多控制器提供直接连接到 USB PHY 的 VBUS 引脚。只要 VBUS 输入超过有效范围，PHY 就会将内部 VBUSVALID 信号置为有效。

该 VBUSVALID 信号非常关键，因为该信号会阻止 USB 器件逻辑在连接有效主机之前启用收发器。如果没有该驱动器，器件可能会错误地尝试驱动总线，从而导致线路上出现未定义的状态或争用。USB 规范规定了一系列去抖间隔，以验证可靠的检测。例如：

- VBUS 升至 4.4V 以上后，器件必须等待至少 100ms，然后才能消耗超过 100mA 的电流。
- 主机在将 D+ 或 D– 上拉电阻解释为有效连接之前，会应用一个去抖周期（至少 100ms）。

这些时序要求可防止由噪声、热插拔事件或电源不稳定引起的错误检测。传统的 USB PHY 集成了这种去抖逻辑，会自动选通内部使能信号，直到条件稳定为止。

传统实现流程：

利用 VBUS 检测的典型 USB 器件初始化序列如下所示：

1. 空闲状态：器件 PHY 已通电，但收发器保持禁用状态。
2. VBUS 检测：主机提供 5V 电源，控制器内部置位 VBUSVALID 信号。
3. 器件使能：USB 控制器在适当的线路（D+ 或 D-）上启用收发器和上拉电阻器。
4. 主机检测器件：主机检测上拉电阻状态，并开始发出复位信号。
5. 枚举：器件响应复位，协商速度，并通过描述符交换开始枚举序列。

此序列会验证是否有序连接以及是否符合 USB 规范。

2.2 AM261x USB 主机检测

对于 AM261x，VBUS 线路不用于配置控制器，如前面几节所述。相反，软件驱动程序会配置控制器的 VBUSVALID 位。若要进行修改，请按照以下各节中讨论的步骤进行操作。

3 硬件修改

默认情况下，TI AM261x LaunchPad 不依赖 VBUS 信号向 USB 控制器发送有关主机存在的信号，因此为了实现真正的主机检测，可以引入简单的硬件修改，以使器件固件可观察到 VBUS 信号。此修改涉及将 5V VBUS 信号路由到 AM261x 上的通用输入/输出 (GPIO) 引脚。器件 I/O 在 3.3V 逻辑电平下运行，因此需要使用电阻分压器来安全地将 5V 输入向下调整至 3.3V 兼容电平。分压器由两个串联的电阻器组成，其连接点输出的电压与电阻的阻值比成比例。在这种情况下，分压器确保 GPIO 引脚在存在 VBUS 时接收到安全的逻辑高电平信号。

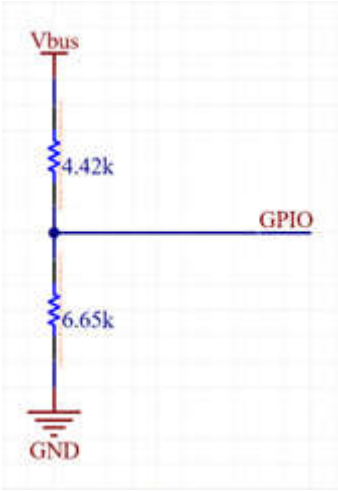

图 3-1. 将 VBUS 电压驱动到 AM261x GPIO 引脚

对于本应用手册中所述的参考实现：

- 已选择 AM261x LaunchPad 上的 GPIO6 用于 VBUS 监控。
- 可以在 J6/J8 BoosterPack 接头的引脚 53 上访问 GPIO6，从而便于使用 GPIO6 与外部分压器电路连接。
- 如 图 3-1 所示，实现了一个简单的分压器电路，其作用是将 USB 的 5V 电压降至大约 3V，以便 AM261x GPIO 将检测为逻辑高电平。
- 路由的 VBUS 信号连接到 GPIO6，该 GPIO6 配置为在上升沿和下降沿均触发中断。
  - 上升沿：指示主机连接事件。
  - 下降沿：指示主机断开事件。

据此，可以修改标准 MCU_PLUS_SDK 软件，使 USB 驱动程序根据 GPIO 输入进行初始化，并仅在检测到主机连接时初始化 USBSS PHY 和控制器，同理，当主机断开连接时，执行 de-init 序列。该方法可确保 AM261x 仅在检测到有效主机时激活 USB 子系统，以防止虚假活动并使器件行为与 USB 规范保持一致。

4 软件修改

在 USB 的 MCU_PLUS_SDK 中，默认的 USB 驱动程序会从 USB 应用程序调用 USB_init() 函数。此函数是从 example.syscfg ti_drivers_open_close.c 文件自动生成的代码调用的。Drivers_usbOpen() 函数。

该函数负责复位 USB dwc_usb3_dev 处理程序、配置 USB PHY 和控制器时钟、设置控制器的控制寄存器、开启 PHY、配置 UTMI OTG 寄存器，并完成 PHY 启动。默认驱动程序不会检测 USB 端口上是否确实存在 USB 连接，而是会对 USB 控制器的 VBUSVALID 位进行编程以强制进行主机连接。这不会对功能产生任何影响。

4.1 USB Synp 驱动程序中的更改

更改 *mcu_plus_sdk/source/usb/synp/soc/* 路径中 device_wrapper.c 文件的默认 USB_init() 函数。USB_init() 函数可以注册 GPIO6 的中断并根据 VBUS 线路的状态，继续进行驱动程序初始化。

```
#include <drivers/gpio.h>
#include <kernel/dpl/AddrTranslateP.h>
#include <kernel/dpl/ClockP.h>

#define HOST_CONNECTED   (1U)
#define HOST_DISCONNECTED   (0U)

uint8_t gHostConnected = false;
uint32_t gGpioBaseAddr = CSL_GPIO0_U_BASE;
uint32_t gHostDetectionPin = 6U; /* GPIO Pin number */
uint32_t gIntrNum = CSLR_R5FSS0_CORE0_INTR_GPIO_INTRXBAR_OUT_14;
HwiP_Object gGpioHwiObject;

void USB_hostDetectGpioIsrFxn(void *args);

void USB_init()
{
    usb_handle.cfg_base = USB_DWC_3;
    usb_handle.dwc_usb3_dev = NULL;
    /* Register GPIO interrupt */
    HwiP_Params     hwiPrms;
    HwiP_Params_init(&hwiPrms);
    hwiPrms.intNum = gIntrNum;
    hwiPrms.callback = &USB_hostDetectGpioIsrFxn;
    hwiPrms.args = (void *)gHostDetectionPin;
    hwiPrms.isPulse = TRUE;
    uint8_t retVal = HwiP_construct(&gGpioHwiObject, &hwiPrms);
    DebugP_assert(retVal == SystemP_SUCCESS );
    /* Check initial state of GPIO - if host is already connected, GPIO will be
high */
    if(GPIO_pinRead(gGpioBaseAddr, gHostDetectionPin) == HOST_CONNECTED)
    {
        gHostConnected = true;
        if (usb_phy_power_sequence() == USB_PHY_OK )
        {
            usbdIntrConfig();
        }
        else
        {
            DebugP_assert(FALSE);
        }
        tusb_init(); /* By default, this is a part of Drivers_usbOpen() in
ti_drivers_open_close.c */
    }
}
```

图 4-1. 在驱动程序中注册 GPIO 中断

```
void USB_hostDetectGpioIsrFxn(void *args)
{
    uint32_t    pinNum = (uint32_t)args;
    uint32_t    bankNum =  GPIO_GET_BANK_INDEX(pinNum);
    uint32_t    intrStatus;

    /* Get and clear bank interrupt status */
    intrStatus = GPIO_getBankIntrStatus(gGpioBaseAddr, bankNum);
    GPIO_clearBankIntrStatus(gGpioBaseAddr, bankNum, intrStatus);

    if(gHostConnected==true && (GPIO_pinRead(gGpioBaseAddr, pinNum) ==
HOST_DISCONNECTED))
    {
        gHostConnected = false;
        /* VBUSVALID. set 0
        */
        HW_WR_FIELD32_RAW(MSS_CTRL + MSS_CTRL_CONTROL_USBOTGHS_CONTROL,
0x00000004,0,0x0);
    }
    else if((GPIO_pinRead(gGpioBaseAddr, pinNum) == HOST_CONNECTED) &&
gHostConnected==false)
    {
        gHostConnected = true;
        if (usb_phy_power_sequence() == USB_PHY_OK )
        {
            usbdIntrConfig();
        }
        else
        {
            DebugP_assert(FALSE);
        }
        tusb_init(); /* By default, this is a part of Drivers_usbOpen() in
ti_drivers_open_close.c */
    }
}
```

图 4-2. 用于主机检测的 GPIO ISR 功能

4.2 USB 应用程序中的更改

首先，从 syscfg 自动生成的代码中禁用 TinyUSB 初始化，仅在检测到主机连接时执行此操作。

要禁用 tusb_init()，请修改 *mcu_plus_sdk/source/sysconfig/usb/.meta/tinyusb/templates/tinyusb_open_close_config.c.xdt* 处的文件，并将其中的“tusb_init()”函数注释掉。相反，将此初始化移动到 USB 驱动程序的初始化序列中。

```
void Drivers_usbOpen(void)
{
    /* initialize USB HW for TI SOC */
    % if(common.getSocName() == "am64x" || common.getSocName() == "am243x") {
    usb_init(&gUsbInitParam);
    % } else {
    USB_init();
    %}
    /* Comment out tiny USB init and move it to the USB driver */
    /* tusb_init(); */

    return;
}
```

图 4-3. 从 Syscfg 模板文件删除 TinyUSB init

现在，在 USB 应用程序中，打开 example.syscfg 并采用以下步骤：

1. 添加 GPIO 引脚。对于此实现，请使用 GPIO6，该引脚也可在 AM261x Launchpad 的 BoosterPack 测试插头上找到，对应 J6 插针的第 53 脚。将触发器类型标记为上升和下降，因为 VBUS 线路在主机连接时被拉高，并在断开时变为低电平，因此预计在这两种情况下都会触发 ISR。

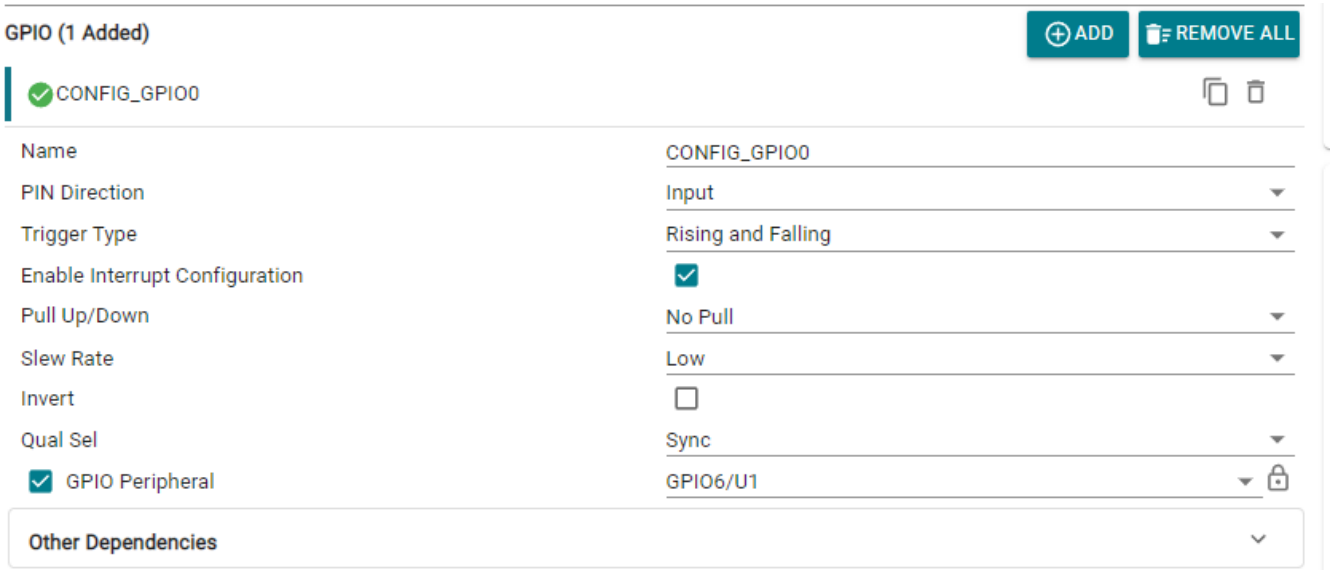

图 4-4. SysCfg GPIO 配置

2. 配置 GPIO XBAR 中断路由。GPIO0-GPIO15 引脚使用 INTR_0 XBAR 输出。因此，对于 GPIO6，配置 INTR_0 和 VIM_MODULE0_0。如果使用了其他一些 GPIO 引脚，请确保正确处理中断配置。

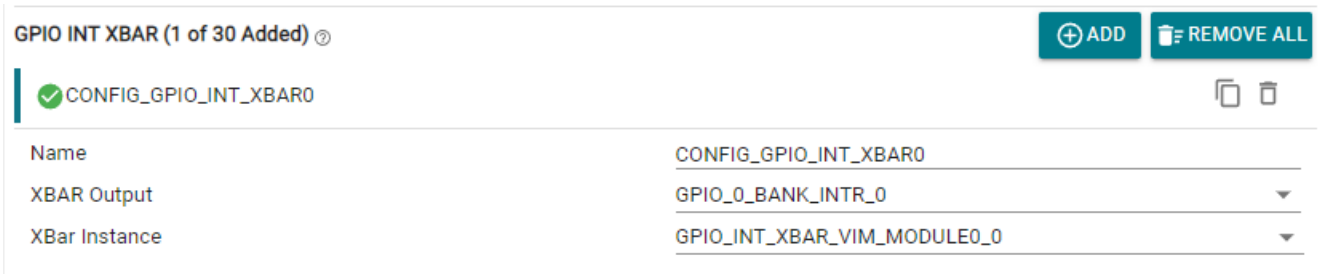

图 4-5. SysCfg GPIO INT XBAR 配置

3. 配置 USB 模块

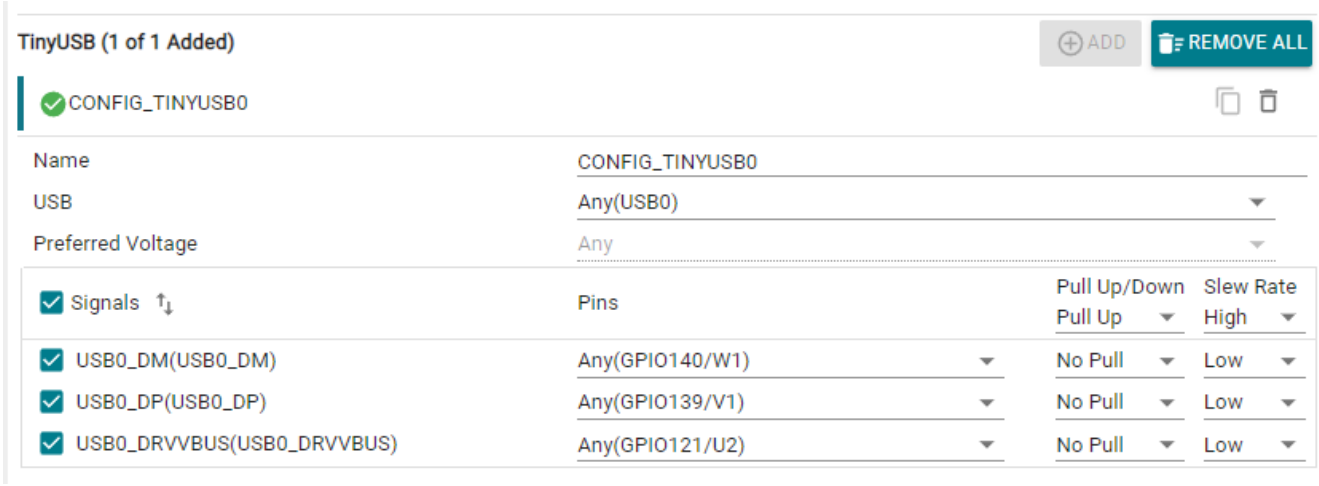

图 4-6. SysCfg USB 配置

4.3 重新构建 USB 驱动程序和应用程序的步骤

已配置驱动程序和应用程序。下一步是重建 USB 库和 USB 应用程序并对其进行测试。

4.3.1 重建 USB 库

MCU_PLUS_SDK 库无法从 CCS 重建。运行以下命令，以从 MCU_PLUS_SDK 目录重新构建 TINYUSB 堆栈和 USB 驱动程序。

```
# re-building NoRTOS USB Lib
gmake -sj -f makefile.am261x usbd_synp_nortos_r5f.ti-arm-clang PROFILE=debug

# re-building FreeRTOS USB Lib
gmake -sj -f makefile.am261x usbd_synp_freertos_r5f.ti-arm-clang PROFILE=debug

# re-building TUSB stack for CDC (NoRTOS)
gmake -sj -f makefile.am261x usbd_tusb_cdc_nortos_r5f.ti-arm-clang PROFILE=debug

# re-building TUSB stack for CDC (FreeRTOS)
gmake -sj -f makefile.am261x usbd_tusb_cdc_freertos_r5f.ti-arm-clang
PROFILE=debug

# re-building TUSB stack for DFU (NoRTOS)
gmake -sj -f makefile.am261x usbd_tusb_dfu_nortos_r5f.ti-arm-clang PROFILE=debug

# re-building TUSB stack for DFU (FreeRTOS)
gmake -sj -f makefile.am261x usbd_tusb_dfu_freertos_r5f.ti-arm-clang
PROFILE=debug

# re-building TUSB stack for NCM (NoRTOS)
gmake -sj -f makefile.am261x usbd_tusb_ncm_nortos_r5f.ti-arm-clang PROFILE=debug

# re-building TUSB stack for NCM (FreeRTOS)
gmake -sj -f makefile.am261x usbd_tusb_ncm_freertos_r5f.ti-arm-clang PROFILE=de-
bug
```

图 4-7. 使用 Makefile 重建 USB 库

4.3.2 重新构建 USB 应用程序

4.3.2.1 CCS 构建

如果应用程序已在 CCS 工作区中，请右键单击该应用程序，然后单击 *Re-build*。此操作会重建并重新链接更新的 USB 驱动程序。

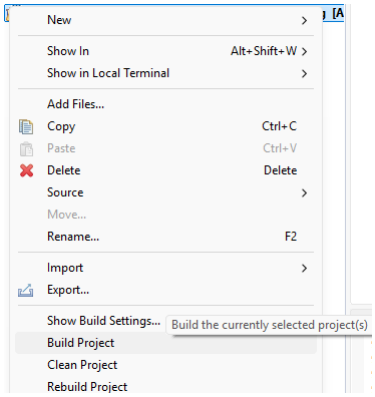

图 4-8. CCS 示例构建步骤

4.3.2.2 命令行构建

若要使用命令行 *make/gmake* 命令构建应用程序，请运行以下命令。

```
gmake -sj -C examples/usb/device/cdc_echo/am261x-lp/r5fss0-0_nortos/ti-arm-clang/
PROFILE=debug all
```

图 4-9. CDC 应用

```
gmake -sj -C examples/usb/device/dfu/am261x-lp/r5fss0-0_nortos/ti-arm-clang/ PRO-
FILE=debug all
```

图 4-10. DFU 应用

```
gmake -sj -C examples/usb/device/ncm/am261x-lp/r5fss0-0_nortos/ti-arm-clang/
PROFILE=debug all
```

图 4-11. NCM 应用

4.4 测试新应用程序

完成所有上述硬件和软件更改后，测试 USB 主机检测现在是否取决于 VBUS 线路。尝试以下方法来验证 USB 是否按预期工作：

1. 使用逻辑分析仪或示波器探测 AM261x-LP 上的 GPIO6 引脚（或使用的任何其他 GPIO），为 LaunchPad 加电并连接/断开电缆几次，以查看 USB VBUS 线路驱动的实际 GPIO 信号。
2. 将 AM261x-LP 配置为适当的引导模式。为了进行测试，TI 建议，使用已将 SBL Null 刷写到 AM261x 器件中的 DEV 引导模式或 OSPI 引导模式。
3. 在 Code Composer Studio 中，启动 USB 应用程序的调试会话，连接到 R5F 内核，并加载 USB 应用程序的调试版本二进制文件。

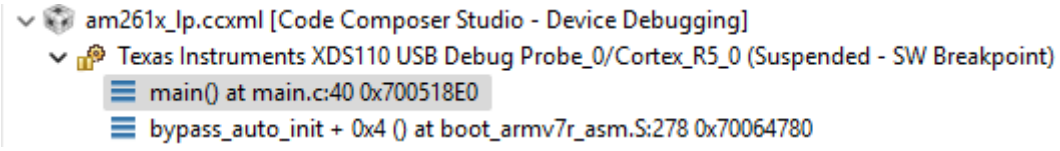

图 4-12. 将应用程序加载到 R5_0 内核

4. 在 USB_hostDetectGpioIsrFxn() 处设置断点，以确保生成中断并触发 ISR。
5. 运行应用程序时，连接并断开 USB 电缆几次，以检查 ISR 是否持续触发。

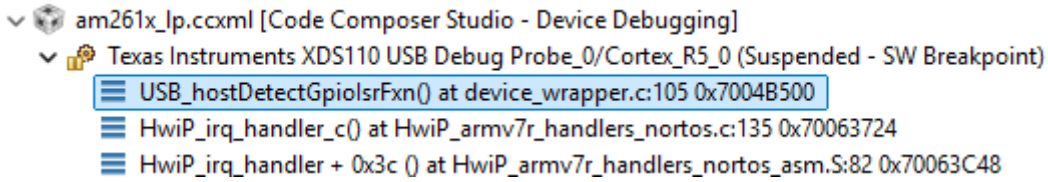

图 4-13. 当主机断开和连接时，应用程序在 GPIO ISR 处停止

6. 每次重新连接 USB 后，检查 USB 传输是否正常。（如果是 CDC 应用程序，请检查 COM 端口通信是否正常工作。如果 DFU 应用程序无法运行，请检查文件传输是否如预期运行）

5 总结

按照前面的步骤，客户可以实现真正可靠的 USB 主机检测，使用 VBUS 电压线检测主机是否存在，从而避免 AM261x USB 出现任何潜在的可靠性或通信问题，并验证主机检测是否真实。

6 参考资料

1. 德州仪器 (TI)，*AM261x Sitara™ 微控制器* 数据表。
2. 德州仪器 (TI)，*AM261x 技术参考手册*，技术参考手册。
3. 德州仪器 (TI)，*AM261x MCU+ SDK 10.02.00* 文档。
4. 德州仪器 (TI)，*AM261x MCU+ SDK 10.02.00* 文档。
5. 德州仪器 (TI)，*AM261x MCU+ SDK USB 示例*，文档。
6. USB，*USB 2.0 规范*文档。
7. 德州仪器 (TI)，*AM2612：AM261x 常见问题解答列表*常见问题解答。

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、与某特定用途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安全、安保法规或其他要求。

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。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 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。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月