Application Note

Implementing VBUS Based USB Host Detection on AM261x

Shaunak Deshpande, Tejas Kulakarni

ABSTRACT

The AM261x Sitara™ Arm® microcontrollers belong to the Sitara AM26x real-time MCU family, designed to address the demanding processing needs of next-generation industrial and automotive embedded systems. Based on scalable Arm® Cortex®-R5F cores, the AM261x delivers high-performance real-time control with an extensive set of peripherals, integrated safety features, and flexible communication interfaces. Among these, USB (Universal Serial Bus) provides a widely adopted, high-speed serial interface for device connectivity.

USB offers a design for connecting a broad range of consumer, industrial, and automotive devices by standardizing power delivery and data transfer between a USB host and a USB device. Correct host detection is fundamental to making sure that a USB device initializes only when a valid host is present, which prevents compliance issues and improving system robustness.

On the AM261x, the USB controller supports operation with an external VBUS voltage. However, the default implementation cannot actively monitor VBUS voltage for host detection. Instead, the USB driver code unconditionally sets the controller registers to power on the PHY and enable the USB module, regardless of whether a host supplying VBUS voltage is present or not. While this approach simplifies initialization, there are some drawbacks such as spurious enumeration attempts, bus contention, increased power consumption, USB specification non-compliance.

This application note demonstrates a reference implementation that modifies the default AM261x USB behavior to incorporate true host detection using VBUS monitoring by using a simple implementation with one GPIO. With this implementation, the device only enables the PHY and initiates enumeration after confirming the presence of a valid host and the VBUS voltage.

The design described in this application note is provided as a reference implementation. This has not undergone extensive USB stress testing on AM261x. Customers are encouraged to adapt, validate, and qualify the approach as appropriate for end applications.

Table of Contents

Trademarks

All trademarks are the property of their respective owners.

1 Introduction

For an in-depth description of the USB subsystem on AM261x devices, see the AM26x MCU+ Academy USB module, the Technical Reference Manual (TRM), and the MCU+ SDK documentation. Figure 1-1 shows the functional block diagram of the USB 2.0 subsystem (USB2SS) on AM261x, which includes a wrapper module, a USB controller module, a PHY module, external interfaces, and internal interfaces.

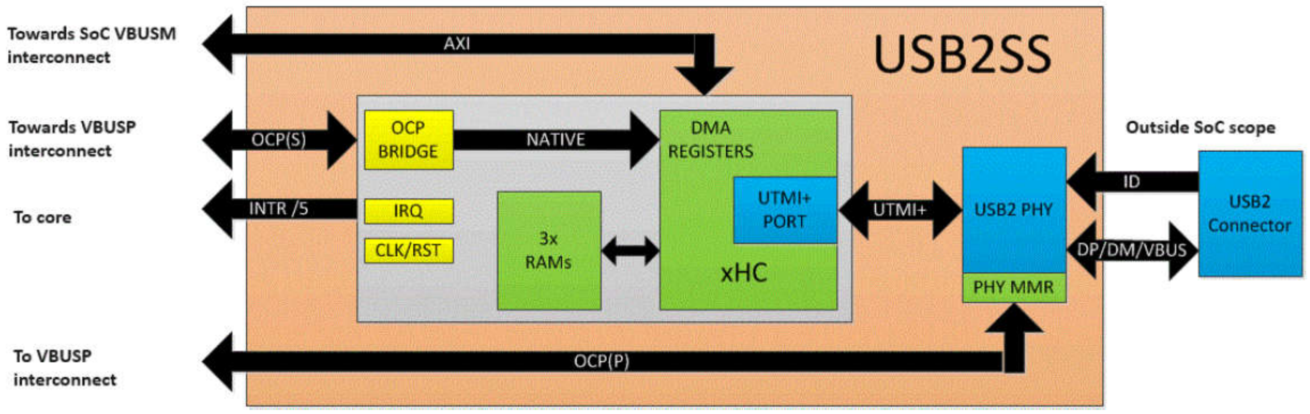


Figure 1-1. AM261x USB SubSystem

The AM261x USB2SS relies on external power logic to generate the 5V VBUS using an external supply. The device provides a dedicated output signal, USB0_DRVVBUS, which is active high and used to enable or disable the external VBUS power source.

When operating in host mode, the USB controller automatically drives USB0_DRVVBUS high, enabling the external charge pump and sourcing 5V on the VBUS line.

When operating in device mode, the controller drives USB0_DRVVBUS low, disabling the external supply and ensuring that the AM261x does not source VBUS.

This control is handled entirely by the USB controller and is transparent to the user, provided that hardware connections and software initialization are configured correctly.

Note that the USB controller is self-powered (through the AM261x device supply pins).
In device mode, the controller does not rely on the externally sourced VBUS voltage
to remain operational. Instead, the presence of VBUS is conveyed internally through the
register: MSS_CTRL.CONTROL_USBOTGHS_CONTROL.CONTROL_USBOTGHS_CONTROL_VBUSVALID (0x50D00894)

This register bit is used to drive the VBUSVALID signal into the USB controller. Since the AM261x does not expose a physical VBUS input pin in device mode, this bit must be configured in software. By default, the AM261x USB driver sets the VBUSVALID bit during USB_init() without verifying the actual presence of a USB host. While this enables the controller and PHY,this bypasses the normal host-detection mechanism defined by the USB specification.

This default behavior introduces several risks:

- Spurious Enumeration Attempts: The device can initiate enumeration even when no host is connected.
- Bus Contention: The device can drive the D+/D– lines without a valid host sourcing VBUS, violating USB protocol rules
- Increased Power Consumption: The PHY remains powered unnecessarily, impacting low-power and battery-sensitive applications
- Non-compliance with USB Specification: The USB standard explicitly requires devices to remain inactive until VBUS is detected.

To verify reliable and spec-compliant operation, the device must implement true host detection based on monitoring the VBUS signal. With proper detection in place, the device only enables the PHY and begins enumeration when a valid host is present, preventing the issues previously outlined.

2 USB Host Detection

2.1 General USB Host Detection Flow

USB (Universal Serial Bus) defines a robust mechanism for detecting when a host and device are connected, establishing the electrical conditions required before any enumeration can take place. In traditional implementations, this detection is primarily achieved through the monitoring of the VBUS line and the D+/D– differential pair, making sure that the controller only enables data communication once a valid host–device relationship is confirmed. The VBUS line is the primary power supply rail provided by the host. In standard USB 2.0 implementations, the host sources 5V on VBUS with a current capability defined by the USB specification (typically 500mA for high-speed USB 2.0, higher for USB 3.x). The device is not expected to drive this line.

Traditional USB device controllers include a VBUS detection circuit, which senses the presence of the 5V signal. The detection is performed using one of two approaches:

- Comparator-based detection: An internal comparator continuously monitors VBUS. Once the voltage rises above a certain threshold (for example., 4.4V minimum defined by the spec), the controller sets a status flag indicating that a valid host connection is present.
- Dedicated VBUSVALID input: Many controllers expose a VBUS pin that connects directly to the USB PHY. The PHY asserts an internal VBUSVALID signal whenever the VBUS input exceeds the valid range.

This VBUSVALID signal is critical, as the signal prevents the USB device logic from enabling the transceivers until a valid host is connected. Without it, the device can erroneously attempt to drive the bus, leading to undefined states or contention on the lines. The USB specification mandates a series of debounce intervals to verify reliable detection. For example:

- After VBUS rises above 4.4V, the device must wait at least 100ms before drawing more than 100mA.
- The host applies a debounce period (at least 100ms) before interpreting a D+ or D– pull-up as a valid connection.

These timing requirements prevent false detection caused by noise, hot-plug events, or unstable power. Traditional USB PHYs integrate this debounce logic, automatically gating internal enable signals until the conditions are stable.

Traditional Implementation Flow:

A typical USB device initialization sequence leveraging VBUS detection proceeds as follows:

1. Idle State: Device PHY is powered but transceivers remain disabled.
2. VBUS Detection: Host supplies 5V, and VBUSVALID is asserted inside the controller.
3. Device Enable: USB controller enables the transceivers and pull-up resistor on the appropriate line (D+ or D–).
4. Host Detects Device: Host senses the pull-up and begins reset signaling.
5. Enumeration: Device responds to reset, negotiates speed, and begins enumeration sequence with descriptor exchange.

This sequence verifies orderly attachment and compliance with the USB specification.

2.2 AM261x USB Host Detection

In case of AM261x, the VBUS line is not used to configure the controller as discussed in the previous sections. Instead the software driver configures the VBUSVALID bit of the controller. To modify this, follow the steps discussed in the following sections.

3 Hardware Modifications

Since the default TI AM261x launchpad does not rely on VBUS signal to signal the USB controller about host presence, to implement true host detection, a simple hardware modification can be introduced to make the VBUS signal observable by the device firmware. The modification involves routing the 5V VBUS signal to a general-purpose input/output (GPIO) pin on the AM261x. Because the device I/O operates at 3.3V logic levels, a resistive voltage divider is required to safely scale the 5V input down to a 3.3V-compatible level. A voltage divider consists of two resistors connected in series, with the junction providing an output voltage proportional to the ratio of the resistor values. In this case, the divider makes sure that the GPIO pin receives a safe logic-high signal when VBUS is present.

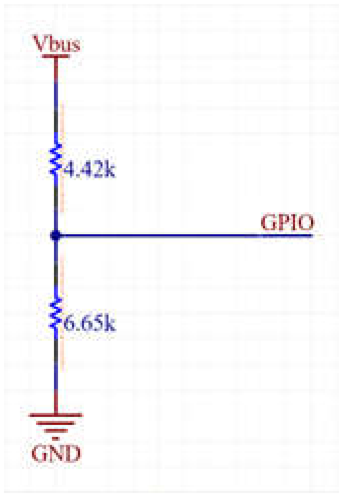


Figure 3-1. Driving the VBUS Voltage to the AM261x GPIO Pin

For the reference implementation described in this application note:

- GPIO6 on the AM261x LaunchPad is selected for VBUS monitoring.
- GPIO6 is accessible on pin 53 of the J6/J8 BoosterPack header, making GPIO6 convenient to interface with the external divider circuit.
- A simple voltage divider circuit is implemented as shown in Figure 3-1, which just brings down the 5V of the USB down to approximately 3V for the AM261x GPIO to detect as a logic high.
- The routed VBUS signal is connected to GPIO6, which is configured to trigger an interrupt on both rising and falling edges.
  - Rising edge: Indicates a host connect event.
  - Falling edge: Indicates a host disconnect event.

Based on this, the standard MCU_PLUS_SDK software can be modified to have the USB driver initialize based on the GPIO input, and initialize the USBSS PHY and controller only when a host connection is detected, and similarly, when the host is disconnected, perform the de-init sequence. This approach makes sure that the AM261x only activates the USB subsystem when a valid host is detected, preventing spurious activity and aligning device behavior with the USB specification.

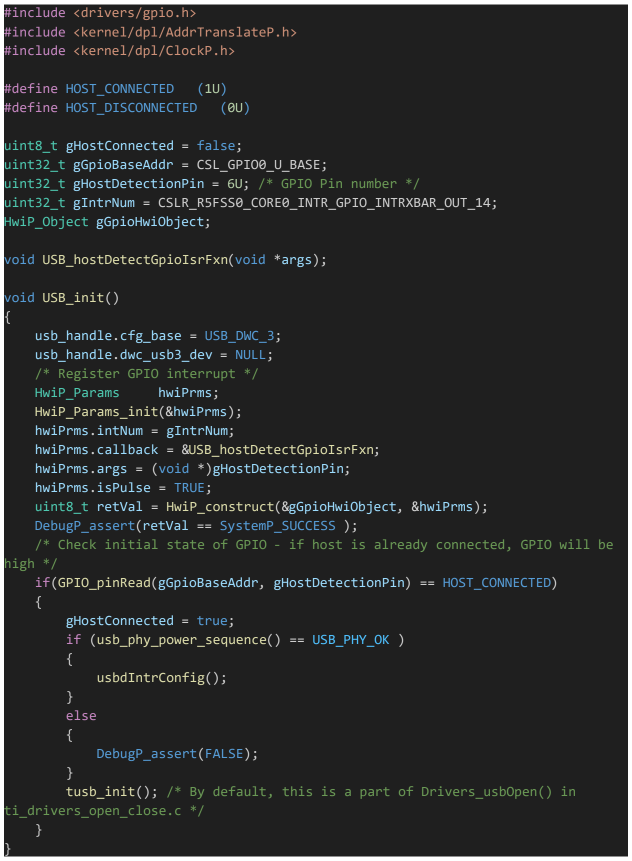
4 Software Modifications

The default USB driver in the MCU_PLUS_SDK for USB calls the USB_init() function from the USB application. This function is called from the code auto-generated from the example.syscfg ti_drivers_open_close.c file. The Drivers_usbOpen() function.

This function is responsible for resetting the USB dwc_usb3_dev handler, configuring the USB PHY and controller clocks, set the control register of the controller, turn on the PHY and configure the UTMI OTG registers and complete the PHY bring up. The default driver does not detect if a USB connection is truly present on the USB port, and programs the VBUSVALID bit of the USB controller to force a host connection. This does not have any impact on functionality.

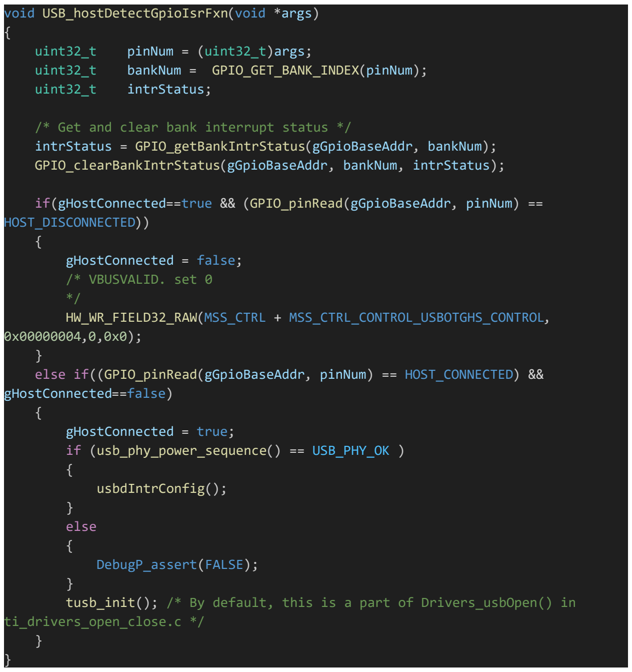
4.1 Changes in the USB Synp Driver

Change the default USB_init() function in the device_wrapper.c file present at *mcu_plus_sdk/source/usb/synp/soc/* path. The USB_init() can function to register an interrupt for GPIO6 and proceed with driver initialization based on the status of the VBUS line.



```
#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 */
    }
}
```

Figure 4-1. Registering GPIO Interrupt in Driver



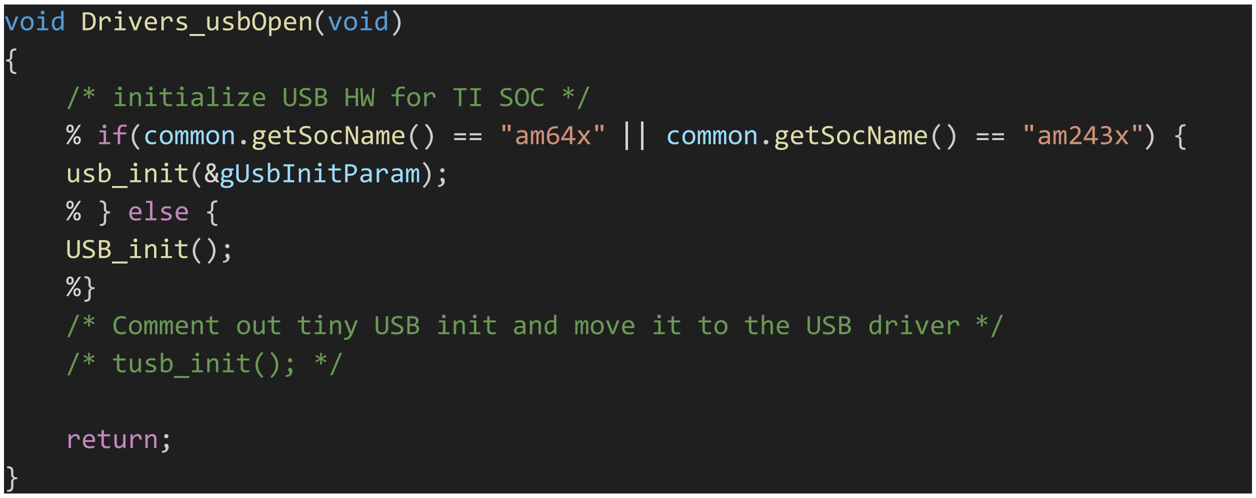
```c
void USB_hostDetectGpioIsrFxn(void *args)
{
    uint32_t    pinNum = (uint32_t)args;
    uint32_t    bankNum =  GPIO_GET_BANK_INDEX(pinNum);
    uint32_t    intrStatus;

    /* 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 */
    }
}
```

Figure 4-2. GPIO ISR Function for Host Detection

4.2 Changes in the USB Application

First, disable the TinyUSB initialization from syscfg auto-generated code, and only do this if the host connection is detected.

To disable tusb_init(), modify the file at *mcu_plus_sdk/source/sysconfig/usb/.meta/tinyusb/templates/tinyusb_open_close_config.c.xdt* and comment out the "tusb_init()" function. Instead, this initialization is moved to the USB driver init sequence.

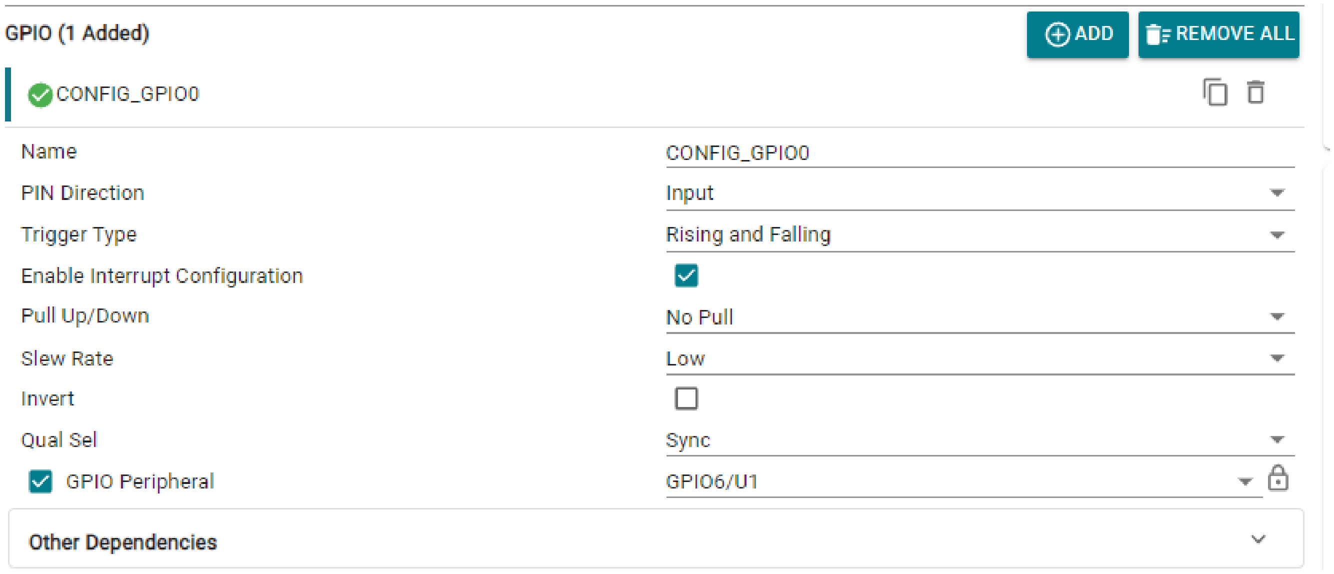


```
void Drivers_usbOpen(void)
{
    /* initialize USB HW for TI SOC */
    % if(common.getSocName() == "am64x" || common.getSocName() == "am243x") {
    usb_init(&gUsbInitParam);
    % } else {
    USB_init();
    %}
    /* Comment out tiny USB init and move it to the USB driver */
    /* tusb_init();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

    return;
}
```

Figure 4-3. Removing TinyUSB init from Syscfg Template File

Now, in the USB application, open the example.syscfg and use the following steps:

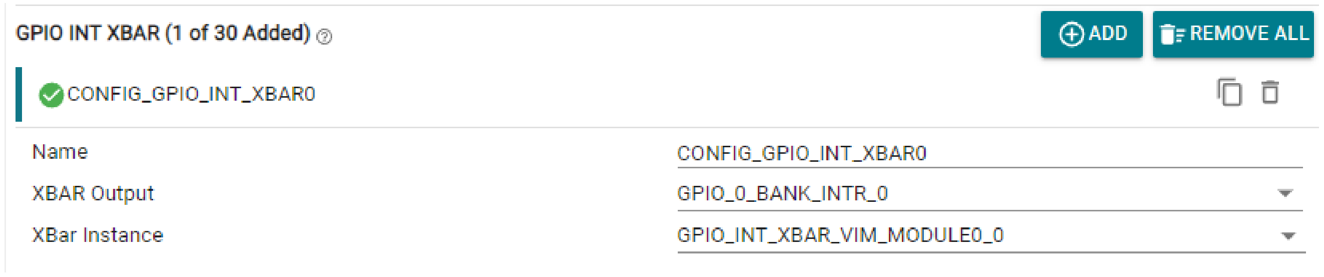
1. Add GPIO pin. For this implementation, use the GPIO6 which is also available on the boosterpack test headers of the AM261x Launchpad as J6 Pin 53. Mark the trigger type as both Rising and Falling since the VBUS line is pulled high on host connect and goes low on disconnect, so the ISR is expected to trigger in both the cases.



Name	CONFIG_GPIO0
PIN Direction	Input
Trigger Type	Rising and Falling
Enable Interrupt Configuration	✓
Pull Up/Down	No Pull
Slew Rate	Low
Invert	☐
Qual Sel	Sync
✓ GPIO Peripheral	GPIO6/U1

Figure 4-4. SysCfg GPIO Configuration

2. Configure the GPIO XBAR interrupt routing. GPIO0-GPIO15 pins use the INTR_0 XBAR Output. So, for GPIO6, configure the INTR_0 and VIM_MODULE0_0. If some other GPIO pin is used, make sure the interrupt configuration is handled properly.



GPIO INT XBAR (1 of 30 Added)

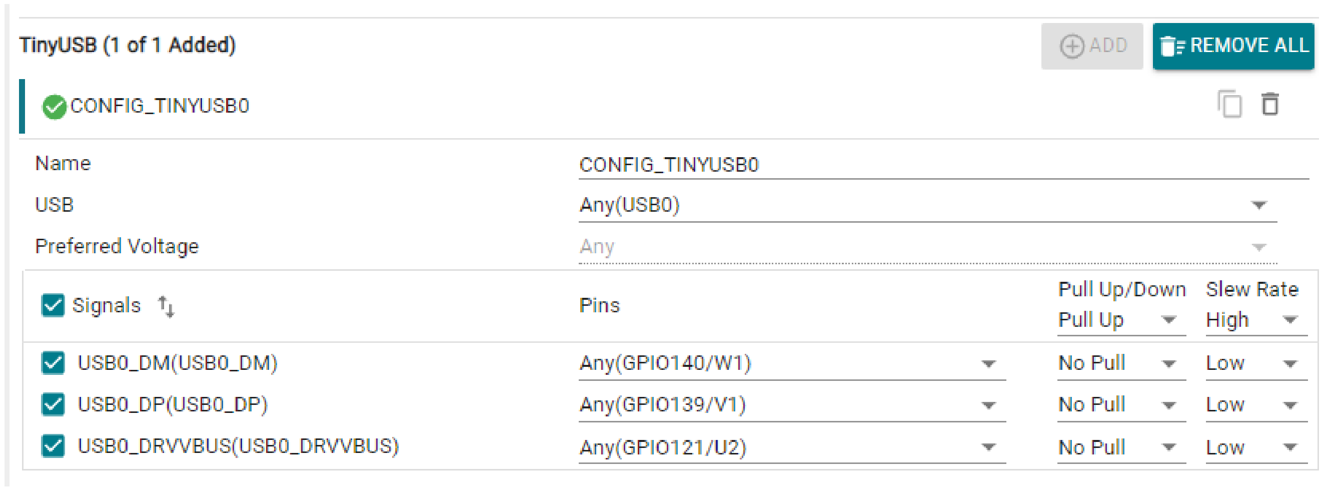
ADD | REMOVE ALL

CONFIG_GPIO_INT_XBAR0

Name	CONFIG_GPIO_INT_XBAR0
XBAR Output	GPIO_0_BANK_INTR_0
XBar Instance	GPIO_INT_XBAR_VIM_MODULE0_0

Figure 4-5. SysCfg GPIO INT XBAR Configuration

3. Configure USB module



TinyUSB (1 of 1 Added)

ADD | REMOVE ALL

CONFIG_TINYUSB0

Name	CONFIG_TINYUSB0
USB	Any(USB0)
Preferred Voltage	Any

✓ Signals	Pins	Pull Up/Down Pull Up	Slew Rate High
✓ USB0_DM(USB0_DM)	Any(GPIO140/W1)	No Pull	Low
✓ USB0_DP(USB0_DP)	Any(GPIO139/V1)	No Pull	Low
✓ USB0_DRVVBUS(USB0_DRVVBUS)	Any(GPIO121/U2)	No Pull	Low

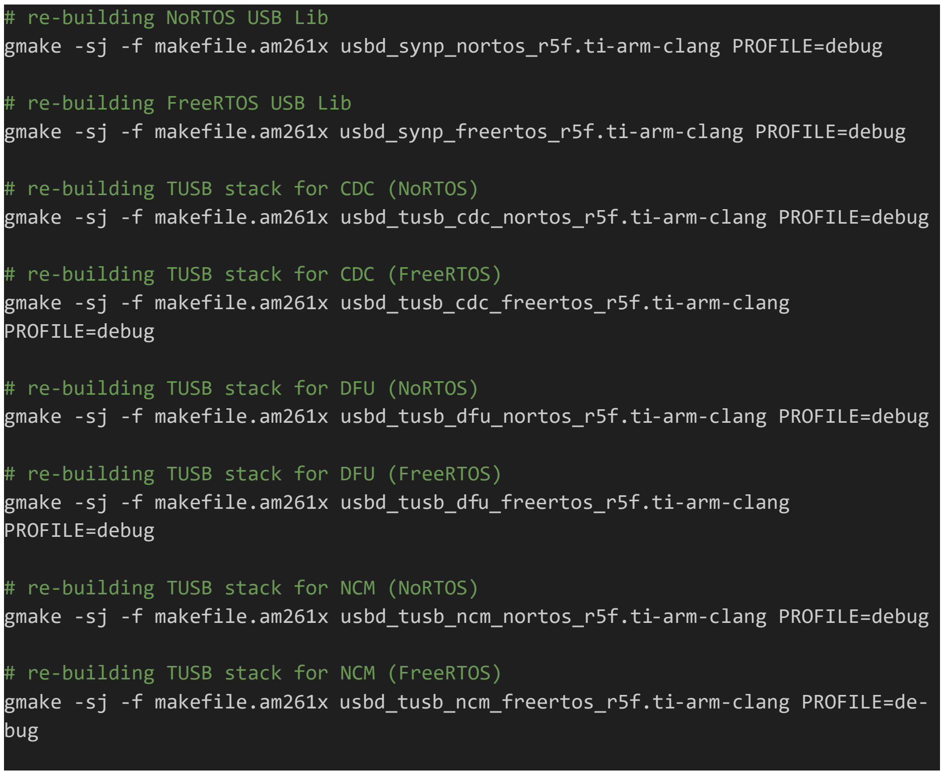
Figure 4-6. SysCfg USB Configuration

4.3 Steps to Rebuild the USB Driver and Application

The driver and application are configured. The next step is to rebuild the USB library and the USB application and test them.

4.3.1 Rebuilding USB Libs

MCU_PLUS_SDK libraries cannot be rebuilt from CCS. Run the following commands to re-build the TINYUSB stack and the USB driver from the MCU_PLUS_SDK directory.



```
# 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
```

Figure 4-7. Rebuilding USB Libraries using Makefiles

4.3.2 Rebuilding USB Application

4.3.2.1 CCS Build

If the application is already in the CCS workplace, right-click on the application and click on *Re-build*. This rebuilds and relinks the updated USB driver.

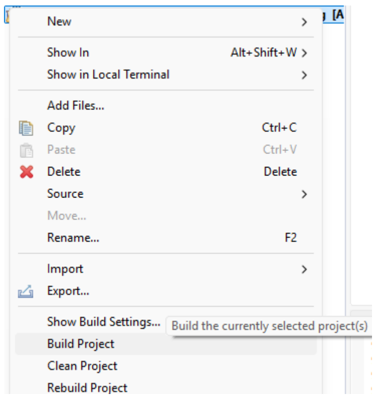


Figure 4-8. CCS Example Build Steps

4.3.2.2 Command Line Build

To build the application using command line *make/gmake* command, run the following command.

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

Figure 4-9. CDC Application

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

Figure 4-10. DFU Application

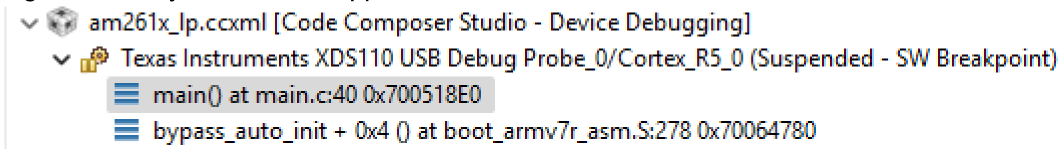
```
gmake -sj -C examples/usb/device/ncm/am261x-lp/r5fss0-0_nortos/ti-arm-clang/
PROFILE=debug all
```

Figure 4-11. NCM Application

4.4 Testing the New Application

After following all the mentioned hardware and software changes, test if the USB host detection is now dependent on the VBUS line. Try the following methods to verify that the USB is working as expected:

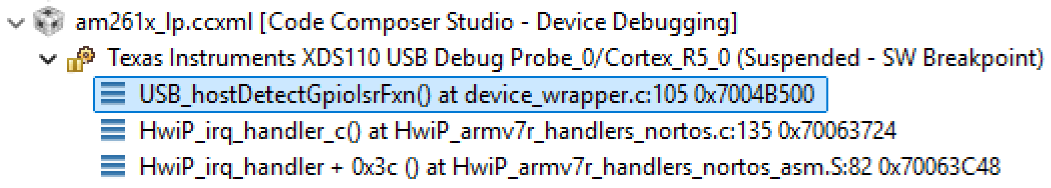
1. Using a logic analyzer or an oscilloscope, probe the GPIO6 pin (or whichever GPIO is used) on the AM261x-LP, power on the Launchpad and connect/disconnect the cable a few times to see the actual GPIO signal being driver by the USB VBUS line.
2. Configure your AM261x-LP in appropriate bootmode. For testing, TI recommends using the DEV Bootmode or OSPI Bootmode with SBL Null flashed to the AM261x device.
3. In Code Composer Studio, launch a debug session for a USB application, connect to the R5F Core and load the debug build binary for the USB application.



```
am261x_lp.ccxml [Code Composer Studio - Device Debugging]
  Texas Instruments XDS110 USB Debug Probe_0/Cortex_R5_0 (Suspended - SW Breakpoint)
    main() at main.c:40 0x700518E0
    bypass_auto_init + 0x4 () at boot_armv7r_asm.S:278 0x70064780
```

Figure 4-12. Loading the Application to R5_0 Core

4. Set a breakpoint at USB_hostDetectGpioIsrFxn() to make sure the Interrupt is generated and the ISR is triggered.
5. While running the application, connect and disconnect the USB cable a few times to check if the ISR is consistently hit.



```
am261x_lp.ccxml [Code Composer Studio - Device Debugging]
  Texas Instruments XDS110 USB Debug Probe_0/Cortex_R5_0 (Suspended - SW Breakpoint)
    USB_hostDetectGpioIsrFxn() at device_wrapper.c:105 0x7004B500
    HwiP_irq_handler_c() at HwiP_armv7r_handlers_nortos.c:135 0x70063724
    HwiP_irq_handler + 0x3c () at HwiP_armv7r_handlers_nortos_asm.S:82 0x70063C48
```

Figure 4-13. Application Halted at the GPIO ISR When Host Disconnects and Connects

6. Every time after reconnecting the USB, check if the USB transfer works. (if CDC application, check if the COM Port communication is functioning. If DFU application does not function, check if the file transfer works as expected)

5 Summary

Following the previous steps, customers can implement true and reliable USB host detection which uses the VBUS voltage line to detect host presence, avoiding any potential reliability or communication issues with AM261x USB and verifying true host detection.

6 References

1. Texas Instruments, *AM261x Sitara™ Microcontrollers*, data sheet.
2. Texas Instruments, *AM261x Technical Reference Manual*, technical reference manual.
3. Texas Instruments, *AM261x MCU+ SDK 10.02.00*, documentation.
4. Texas Instruments, *AM261x MCU+ SDK 10.02.00*, documentation.
5. Texas Instruments, *AM261x MCU+ SDK USB examples*, documentation.
6. USB, *USB 2.0 Specification*, documentation.
7. Texas Instruments, *AM2612: List of FAQs for AM261x*, FAQs.