APPLICATION NOTE

AT06475: SAM D21 USB

Atmel SAM D21

Introduction

SAM D21 USB peripheral supports both embedded host and device modes of operation, supporting full speed (12Mbits/s) and low speed (1.5Mbits/s) communication.

This application note complements the information provided in the SAM D21 datasheet and provides a detailed view on USB peripheral requirements and configuration options for using with ASF USB Stack and SAM D21 USB driver.

Features

- USB 2.1 Specification Compliance with Link Power Management (LPM-L1) Protocol Support
- Embedded host and device mode support
- Full Speed and Low Speed Support
- Built-in DMA for data transfers
- Multi-packet transfers
- Ping-pong mode
- On-chip USB transceiver
- On-chip pull-up and pull-down resistors
- On-chip USB serial resistors

Table of Contents

1. SAM D21 USB Peripheral Overview 3
   - 1.1 Introduction 3
   - 1.2 Common Features for Host and Device Modes 3
     - 1.2.1 Multi-packet Transfers 3
     - 1.2.2 Ping-pong Operation 4
   - 1.3 USB Device Mode 4
     - 1.3.1 USB Device Mode Features 4
     - 1.3.2 Crystal-less Operation 5
     - 1.3.3 Endpoint Management 5
     - 1.3.4 Suspend and Remote Wake-up 5
     - 1.3.5 Link Power Management Protocol 5
     - 1.3.6 SOF Clock Output 6
     - 1.3.7 Feedback Endpoint 6
   - 1.4 USB Embedded Host Mode 6
     - 1.4.1 USB Embedded Host Mode Features 6
     - 1.4.2 Pipe Configuration 6
     - 1.4.3 Suspend and Remote Wake-up 7
     - 1.4.4 Link Power Management Protocol 7
     - 1.4.5 Phase Locked SOF 7
     - 1.4.6 Multiplexed Virtual Pipes 7
2. SAM D21 USB Application Configuration 8
   - 2.1 USB Connection 8
     - 2.1.1 USB I/O Pins 8
     - 2.1.2 USB VBUS and ID pin Management 8
     - 2.1.3 Bus-Powered USB Device Application 9
     - 2.1.4 Self-Powered USB Device Application 9
     - 2.1.5 USB Embedded Host Application 10
     - 2.1.6 USB Dual Role Application 10
   - 2.2 USB Clock Configuration 11
     - 2.2.1 Clock Parameters used in USB Driver 11
     - 2.2.2 GCLK Generator for GCLK_USB 11
     - 2.2.3 DFLL as Clock Source 12
       - 2.2.3.1 DFLL USBCRM Mode (Crystal-less USB operation) 12
       - 2.2.3.2 DFLL Closed Loop Mode 12
     - 2.2.4 DPLL as Clock Source 13
   - 2.3 Low Power Management with SAM D21 USB 14
     - 2.3.1 Sleep Modes with USB Stack 14
     - 2.3.2 Link Power Management (LPM-L1) Protocol for USB Device 15
     - 2.3.3 Link Power Management (LPM-L1) Protocol for USB Host 16
   - 2.4 SAM D21 USB Features Support in the USB driver 17
3. Atmel USB Stack v2 18
4. References 19
   - 4.1 Device datasheet 19
   - 4.2 ARM documentation on Cortex-M0+ core 19
   - 4.3 Atmel Studio 19
5. Revision History 20

1. SAM D21 USB Peripheral Overview

1.1 Introduction

The USB peripheral supports both device and embedded host modes having eight endpoints/pipes (each address having one input and one output endpoint).

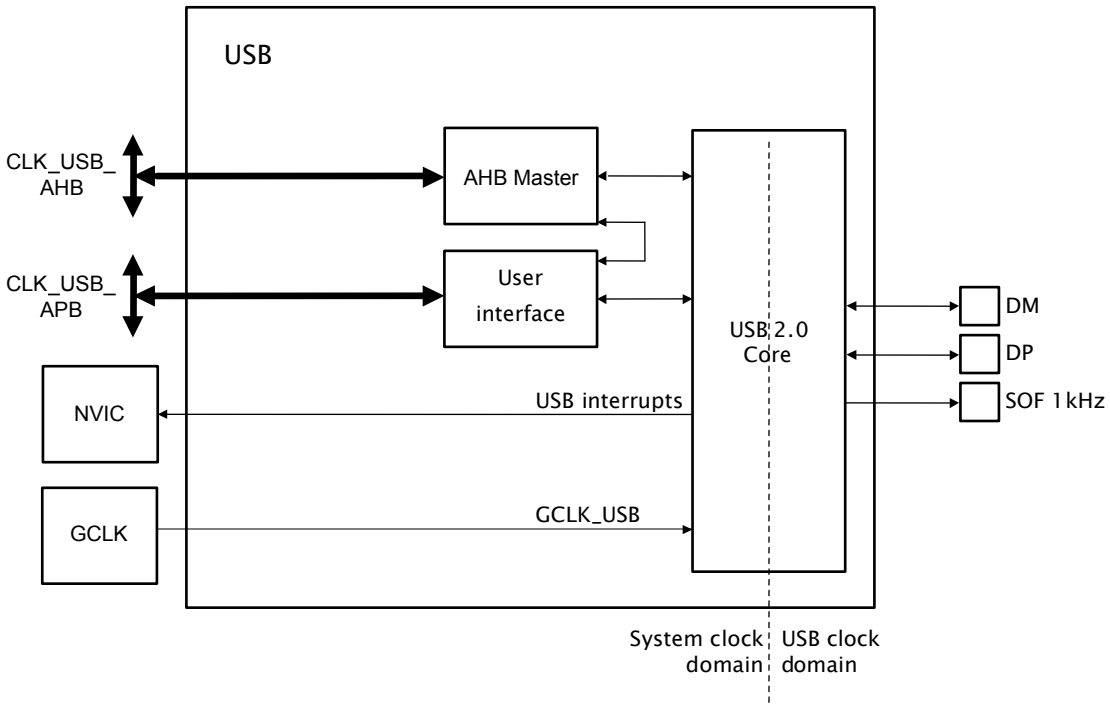
To optimize the endpoint/pipe data transfers, the USB peripheral features a built-in DMA which will read/write from/to the internal SRAM during a USB transaction without CPU intervention.

In addition, the USB peripheral also has support for ping-pong operation and multi-packet transfers. This reduces the number of software interventions and interrupts required to manage a USB transaction.

It is not possible to configure the device and host registers at the same time, as they share the same address space for their registers. So, the USB peripheral can be configured in only one of the modes at any given point of time.

SAM D21 USB uses CLK_USB_APB for register access and CLK_USB_AHB for the DMA access to internal SRAM. It uses GCLK_USB as reference clock. Figure 1-1 shows the USB peripheral block diagram.

Figure 1-1. SAM D21 USB Peripheral

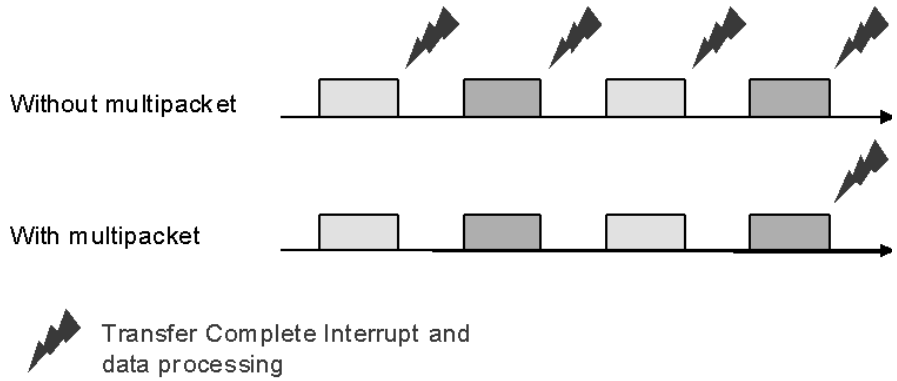


1.2 Common Features for Host and Device Modes

1.2.1 Multi-packet Transfers

Typically, an endpoint can transfer a maximum of endpoint packet size in a single USB transfer. For large amount of data, the software has to manually split the data into packets and send them through the USB. When multi-packet transfer is used, the USB peripheral itself splits the data into multiple packets and transfers them automatically on each USB data request. Thus, it enables transfer of data larger than the endpoint size without further interrupts or CPU intervention as shown in Figure 1-2 and provides higher data transfer rate.

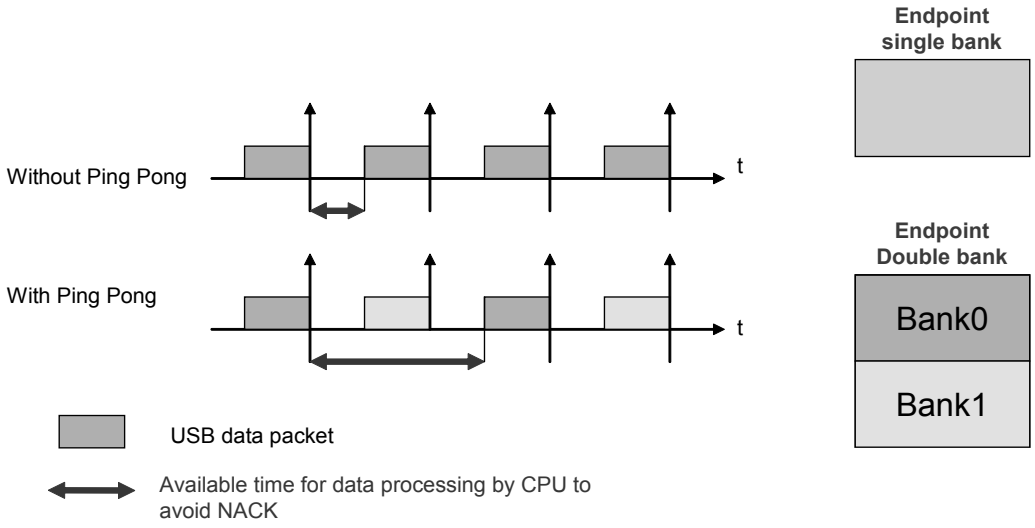
Figure 1-2. USB Transfer with Multi-packet



1.2.2 Ping-pong Operation

When Ping-Pong mode/dual-bank is configured for an endpoint/pipe, it uses the data buffers for both the input and output endpoints/pipes of an endpoint/pipe address in the same direction. For example, if Endpoint 1IN is configured to use ping-pong mode, it uses the data buffer of Endpoint 1 OUT for dual-banking. (Endpoint 1 OUT cannot be used when Endpoint 1 IN is used in ping-pong mode). An example for Ping-Pong operation is shown in Figure 1-3.

Figure 1-3. Ping-Pong Operation in Endpoint/Pipe



1.3 USB Device Mode

1.3.1 USB Device Mode Features

SAM D21 USB peripheral has following features specific to USB device mode:

- USB clock recovery mode for DFLL in USB device mode (Crystal-less USB operation)
- Feedback Endpoint Support
- No endpoint size limitation

1.3.2 Crystal-less Operation

SAM D21 provides an option to use the USB SOF (Start-Of-Frame) signal as a reference signal for DFLL to generate the required 48MHz for GCLK_USB. When the USB Clock Recovery Mode (USBCRM) is enabled in the DFLL, the SOF signal from the USB host will be used as the reference clock for DFLL. An auto jitter mechanism is enabled with USBCRM to manage the jitter within the USB specification limit. This feature eliminates the requirement of an external crystal in USB device applications.

1.3.3 Endpoint Management

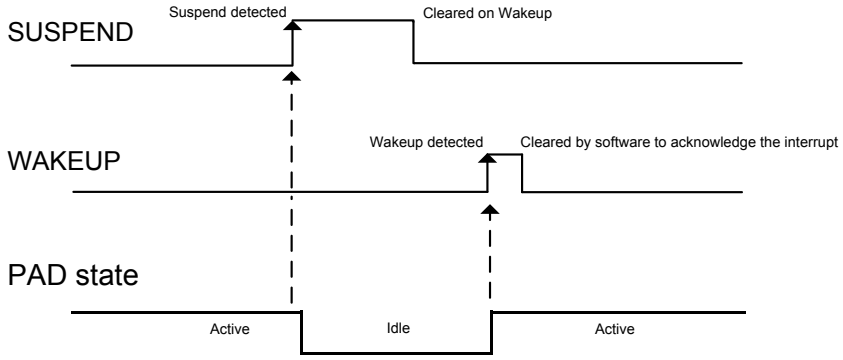
The SAM D21 USB device mode supports a maximum of eight endpoint addresses. Each address can be configured to have an input and an output endpoint. All the endpoints can be configured in any of the four transfer types: control, interrupt, bulk or isochronous.

SAM D21 uses an endpoint descriptor table to manage the endpoint configuration and status. The endpoint descriptor and the endpoint data buffer can be physically allocated anywhere in the internal RAM. The USB controller accesses the endpoint data directly through the AHB master with the help of the built-in DMA. Memory usage for each endpoint depends on the endpoint configuration (size, number of banks, etc) and can be allocated dynamically by the user based on the application requirements.

1.3.4 Suspend and Remote Wake-up

When a suspend condition is detected on the USB bus, the USB pads are put into idle state (low power consumption mode). On a resume signal from the host, the pads are back to the active state.

Figure 1-4. Pad Behavior on Suspend and Resume Event



The USB operations will resume on receiving a downstream signal from the host. If remote wake-up is enabled in the device, it can send an Upstream Resume to the host to resume the USB operations. The rebroadcasted resume from the USB host is managed by the USB hardware and sets an interrupt flag when the resume event is completed. Figure 1-4 shows the USB pad behavior on suspend and resume events.

1.3.5 Link Power Management Protocol

SAM D21 USB device supports the Link Power Management Protocol (LPM-L1) and complies with the USB 2.0 LPM-L1 ECN.

LPM-L1 allows a USB host to configure the USB device into inactive state much faster than the normal USB suspend mode (which requires 3ms of bus inactivity). It also provides much faster wake-up times in the order of micro-seconds compared to the generic resume by host or upstream resume by device (which requires nearly 3 to 30ms). Further, it imposes no restrictions on the current drawn from the VBUS compared to SUSPEND mode (which is limited to 500µA to 2mA).

A LPM transaction from host to device is necessary for the device to enter the L1 state. This L1-SLEEP state would occur after 9µs from receiving the LPM transaction. The remote wake-up feature from the device can be enabled/disabled through the LPM transaction. The L1-SLEEP exit time is specified by the BESL parameter provided through the USB descriptors. The device can conserve power in L1-SLEEP state by entering any of the sleep modes with the constraint that the wake-up latency should satisfy the BESL parameter.

If the LPM-L1 support is not enabled, the device will ignore the LPM transactions from the USB host and no handshake is returned.

1.3.6 SOF Clock Output

The USB SOF signal can be brought out to the SOF 1kHz pin. That way it can be fed to external components as a reference/synchronization signal.

1.3.7 Feedback Endpoint

SAM D21 USB supports the creation of explicit feedback endpoints. Feedback endpoints are generally used in asynchronous USB Audio applications to provide an explicit feedback on the isochronous endpoints usage in the device. It provides a mechanism to the host on adjusting the data rate based on the device operation. A feedback endpoint is an interrupt endpoint having the same endpoint number as the isochronous endpoint but operating in the opposite direction.

Example: Endpoint Address 1 OUT: Isochronous (Audio data from host to device).
Endpoint Address 1 IN: Interrupt (Provides feedback on the audio data usage to host).

1.4 USB Embedded Host Mode

1.4.1 USB Embedded Host Mode Features

SAM D21 USB peripheral has following features specific to USB embedded host mode:

- Multiplexed virtual pipe Support
- Phase-locked SOF Support
- No pipe size limitation
- Feedback endpoint support

1.4.2 Pipe Configuration

The SAM D21 USB host mode supports a maximum of eight physical pipe addresses. Each address can be configured to have an input and an output pipe. There is no limitation for the pipe size. All the pipes can be configured in any of the four transfer types: control, interrupt, bulk or isochronous. In addition to this functionality, the SAM D21 USB provides the feature to multiplex virtual pipes to a single physical pipe.

SAM D21 uses a pipe descriptor table to manage the pipe configuration and status. The pipe descriptor and the pipe data buffer can be physically allocated anywhere in the internal RAM. The USB controller accesses the pipe data directly through the AHB master with the help of the built-in DMA. Pipes can be configured to use a maximum of two memory banks.

For IN/OUT transactions, the USB will send the IN/OUT requests for the pipe until it is frozen. When the pipe is configured to use multiple banks, the USB peripheral will automatically switch to the next bank (the next bank must be ready) for IN/OUT transactions.

The pipes can be frozen by the hardware in one of the following cases:

- After the pipe is enabled
- On receiving a STALL handshake
- On completing a transfer with a pipe error
- On the completion of a LPM transaction (on receiving or handshake or on time-out)

If the software tries to freeze a pipe which is in the middle of a USB transaction, the USB will complete the ongoing USB transaction before freezing the pipe.

1.4.3 Suspend and Remote Wake-up

On disabling the SOF generation, the USB host puts the bus into idle state. After 3ms of inactivity in the bus, the connected USB device would enter the Suspend Mode as per the USB specifications.

While enabling the Suspend state, if a bank is ready for OUT transaction, the USB will automatically exit the Suspend state to start the OUT transaction.

The USB host can be restored to its active state by sending a USB Resume to the device or on receiving an Upstream Resume from the device.

1.4.4 Link Power Management Protocol

SAM D21 USB host supports the Link Power Management Protocol and complies with the USB 2.0 LPM-L1 ECN. It supports the hardware generation of LPM transactions required to enter the L1-SLEEP state. The USB host will enter the L1-SLEEP state only on receiving a successful handshake from the device. The host will cease the generation of the SOF in the L1-SLEEP state. The host exits the L1 state on receiving an L1 Upstream Resume from device or by generating L1 Resume or a Downstream Resume. If it is an L1 Resume, the timing is provided by the BESL parameter. After resuming from the L1-SLEEP state, the host starts Start-Of-Frame generation.

1.4.5 Phase Locked SOF

SAM D21 USB host generates phase locked SOF which are typically used by Synchronous Endpoints in USB Audio applications where the audio stream is synchronized with the USB SOF signal. Here, the SOFs will be generated every 1ms or an integer multiple of 1ms (across idle/suspend/resume state) to maintain the synchronicity for synchronous endpoints.

1.4.6 Multiplexed Virtual Pipes

SAM D21 USB allows the mapping of one physical pipe to different logical pipes creating a virtual pipe configuration. To implement a virtual pipe for multiple logical pipes, context saving (Pipe registers and Pipe configuration data in descriptor table) and switching of pipe configurations should be done before using the logical pipe.

2. SAM D21 USB Application Configuration

The following sections describe the USB I/O connections, VBUS and ID pin monitoring, USB clock requirements and options, Sleep modes integrated with the USB driver and LPM mode support in USB device and host for a SAM D21 USB Application.

2.1 USB Connection

2.1.1 USB I/O Pins

Table 2-1. USB I/O Lines

Signal	I/O Pin	I/O Peripheral Mux	Description
USB/DM	PA24	G	USB Data Negative line
USB/DP	PA25	G	USB Data Positive line
USB/SOF 1kHz	PA23	G	USB SOF Output

By default, when udc_start() is called, the USB stack initializes the pins PA24 and PA25 to USB/DM and USB/DP peripheral functionality respectively. Hence, these pins cannot be used for any other functions than USB after initializing the USB stack.

USB/SOF 1kHz can optionally be enabled based on application requirements.

Code snippet to enable the USB/SOF 1kHz peripheral functionality in pin PA23

```
struct system_pinmux_config pin_config;
system_pinmux_get_config_defaults(&pin_config);
pin_config.mux_position = MUX_PA23G_USB_SOF_1KHZ;
system_pinmux_pin_set_config(PIN_PA23G_USB_SOF_1KHZ, &pin_config);
```

This USB SOF 1kHz output can be used to synchronize external components with the USB Start-of-Frame signal. A typical example would be a Synchronous USB Audio Application. An external clock synthesizer uses this USB SOF 1kHz output signal as a reference input to generate a USB synchronized clock for the audio codec.

2.1.2 USB VBUS and ID pin Management

SAM D21 USB peripheral does not have dedicated pins for VBUS and ID. By default, the USB driver is configured to use only External Interrupt Controller (EIC) pins for VBUS and ID.

The board header should contain the mapping for VBUS and ID pins.

For example, in SAM D21 Xplained Pro, the board header samd21_xplained_pro.h has

```
#define USB_VBUS_PIN PIN_PA14
#define USB_ID_PIN   PIN_PA03
```

To enable the VBUS and ID pin management in the USB stack, the following configuration has to be added in conf_board.h in the application

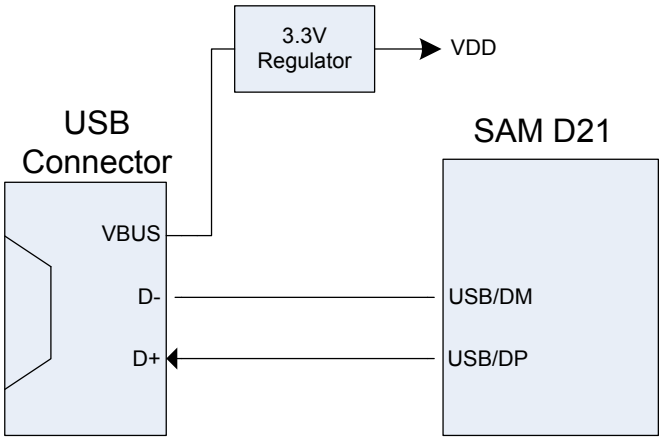
```
#define CONF_BOARD_USB_VBUS_DETECT
#define CONF_BOARD_USB_ID_DETECT
```

The above configurations are checked in the USB driver during compile time for adding the support for VBUS and ID pins. External interrupt peripheral functionality is enabled on the GPIO pins configured for VBUS and ID.

2.1.3 Bus-Powered USB Device Application

In a bus-powered USB device application, VBUS monitoring is not required, as the device is going to be powered through the VBUS pin. In device-only applications, ID pin monitoring is also not required. The 5V VBUS is connected to a 3.3V regulator to supply the device (VDD) as shown in Figure 2-1.

Figure 2-1. SAM D21 Bus Powered USB Device Application

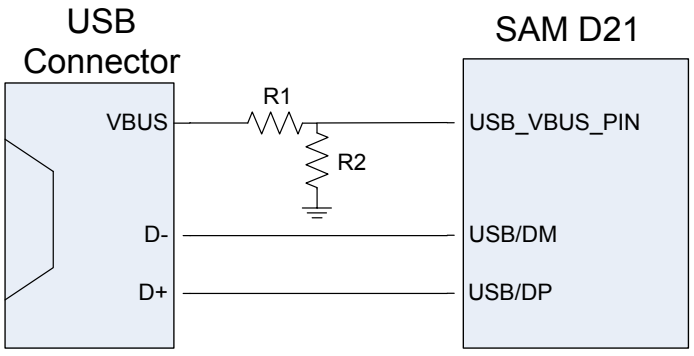


2.1.4 Self-Powered USB Device Application

In a self-powered USB device application, VBUS monitoring is optional. In device-only applications, ID pin monitoring is also not required.

VBUS monitoring can be used in application which has the requirement to detect the presence of a USB host to perform any additional operations. In order to monitor the VBUS, it should be connected to an EIC pin configured as input. The USB driver initializes the EIC interrupt handler for the VBUS pin. The EIC module triggers an interrupt when it detects a change in the VBUS pin state. SAM D21 IO pins support a maximum of 3.3V input. Hence, to connect to 5V VBUS, a voltage divider circuit has to be implemented to limit the input voltage to the GPIO pin as shown in Figure 2-2.

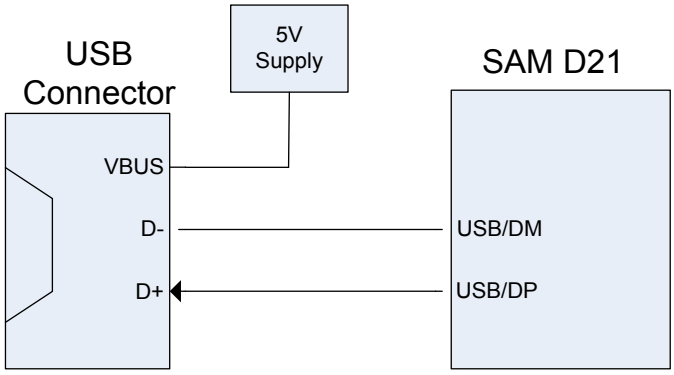
Figure 2-2. SAM D21 Self-Powered USB Device Application



2.1.5 USB Embedded Host Application

In embedded host operation, since there is no dedicated VBUS supply pin in SAM D21, the connected USB device has to be powered through an external power source. In embedded host-only applications, ID pin monitoring is not required. The VBUS pin in the USB connector should be connected to an external power source as shown in Figure 2-3.

Figure 2-3. SAM D21 USB Host Application



2.1.6 USB Dual Role Application

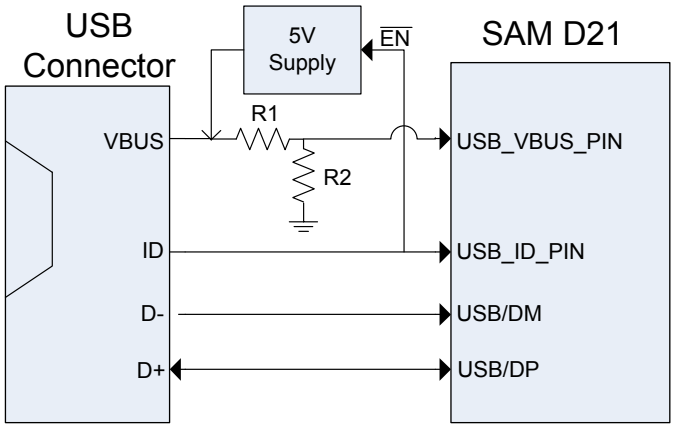
In a dual role application, where SAM D21 can operate in either device or host mode as per the ID pin, the ID pin can be used to control the external supply output. The USB_ID_PIN input is required to identify the mode of operation to initialize the USB peripheral in host or device mode. The USB_VBUS_PIN input is optional and be used only in the device mode to identify the presence of a USB host.

In order to monitor VBUS and ID, they should be connected to EIC pins configured as input. The ID pin is configured with an internal pull-up. The USB driver initializes the EIC interrupt handler for the VBUS and ID pins. The EIC module triggers an interrupt when it detects a change in the VBUS ad ID pin states.

SAM D21 IO pins support a maximum of 3.3V input. Hence, to connect the 5V VBUS, a voltage divider circuit has to be implemented to limit the input voltage to the GPIO pin as shown in Figure 2-4.

When a USB Micro-A plug (USB Embedded host mode) is connected, the ID pin will be at low level enabling the external supply for sourcing VBUS. When a USB Micro-B plug (USB device mode) is connected, the ID pin will be left floating. Since the internal pull-up is enabled by the USB driver, it will be pulled high disabling the external supply on VBUS pin. As per the USB specifications, an USB device should not back power the VBUS pin.

Figure 2-4. SAM D21 USB Dual Role Application



2.2 USB Clock Configuration

SAM D21 USB requires a 48MHz ±0.25% reference clock (GCLK_USB) for its operations. This GCLK_USB is used to generate the required bit clock from the received USB differential data.

In addition to the GCLK_USB requirement, the CLK_USB_AHB should be a minimum of 8MHz to follow the USB data rate in full-speed mode.

2.2.1 Clock Parameters used in USB Driver

The following table provides the clock configuration parameters used in conf_usb.h of the SAM D21 USB driver.

Table 2-2. USB Driver Clock Configuration Parameters

Parameters	Default Value	Description
UDD_CLOCK_GEN	GCLK_GENERATOR_0	GCLK Generator used as source for GCLK_USB
UDD_CLOCK_SOURCE	SYSTEM_CLOCK_DFLL	Clock source used in the GCLK generator to generate GCLK_USB

The USB driver uses default configuration of the DFLL as the clock source for GCLK_GENERATOR_0 to generate the GCLK_USB.

Note: The UDD_CLOCK_SOURCE does not initialize and start the required clock source. It is added to track the clock source status (clock ready or not) when enabling the GCLK_USB. The actual clock configurations are managed through the conf_clocks.h configuration header.

2.2.2 GCLK Generator for GCLK_USB

By default, GCLK_GENERATOR_0 is used for generating GCLK_USB by the USB driver. Note that in the default configuration GCLK_GENERATOR_0 also acts as the CPU clock. If different USB and CPU clocks are needed, then the GCLK generator used for USB has to be changed. There is no restriction in choosing the GCLK generator used to generate GCLK_USB.

GCLK_USB can be generated from one of the following clock sources in USB device mode.

- DFLL with USB Clock Recovery Mode (Crystal-less operation)
- DFLL with external crystal/external clock in XOSC32K/XIN
- DPLL with external crystal/external clock in XOSC32K/XIN

GCLK_USB can be generated from one of the following clock sources in USB Embedded host mode.

- DFLL with external crystal/external clock in XOSC32K/XIN
- DPLL with external crystal/external clock in XOSC32K/XIN

Example Configuration: Assign GCLK_GENERATOR_1 to generate GCLK_USB

In conf_clocks.h, modify the source for the GCLK generator used to generate GCLK_USB (for e.g., if GCLK_GENERATOR_1 should be used to generate GCLK_USB), then:

```
#define CONF_CLOCK_GCLK_1_CLOCK_SOURCE          SYSTEM_CLOCK_SOURCE_DFLL (or)
#define CONF_CLOCK_GCLK_1_CLOCK_SOURCE          SYSTEM_CLOCK_SOURCE_DPLL
```

Configure the other parameters for GCLK_GENERATOR_1 in conf_clocks.h as per application requirements.

```
/* Enable GCLK generator 1 */
#define CONF_CLOCK_GCLK_1_ENABLE                true
/* Run in Standby mode configuration for GCLK generator 1 */
#define CONF_CLOCK_GCLK_1_RUN_IN_STANDBY        true
/* Divider for GCLK generator 1 output */
```

```
#define CONF_CLOCK_GCLK_1_PRESCALER             1
/* Enable/Disable GCLK generator 1 output in GCLK pin */
#define CONF_CLOCK_GCLK_1_OUTPUT_ENABLE         false
```

Add a define in conf_usb.h to override the default GCLK generator used in USB driver.

```
#define UDD_CLOCK_GEN GCLK_GENERATOR_1
```

In conf_usb.h, add a define to override the default clock source checked in USB driver.

```
/* When DFLL is used as clock source for GCLK generator */
#define UDD_CLOCK_SOURCE SYSTEM_CLOCK_SOURCE_DFLL (or)
/* When DPLL is used as clock source for GCLK generator */
#define UDD_CLOCK_SOURCE SYSTEM_CLOCK_SOURCE_DPLL
```

If phase-locked SOF feature is being used in USB embedded host application, then the GCLK_USB should not be stopped in USB Suspend mode where the SOF generation is suspended.

2.2.3 DFLL as Clock Source

DFLL can be used as clock source in two modes:

1. USBCRM mode (Crystal-less operation - Applicable only for USB device mode).
2. Closed loop mode with external crystal.

2.2.3.1 DFLL USBCRM Mode (Crystal-less USB operation)

When the USB Clock Recovery Mode (USBCRM) is enabled in the DFLL, the SOF signal from the USB host will be used as the reference clock for DFLL. An auto jitter mechanism is enabled with USBCRM to manage the jitter within the USB specification limit. This eliminates the requirement of an external crystal in USB device applications.

Example Configuration: DFLL USBCRM Mode in conf_clocks.h

```
/* SYSTEM_CLOCK_SOURCE_DFLL configuration - Digital Frequency Locked Loop */
#define CONF_CLOCK_DFLL_ENABLE            true
#define CONF_CLOCK_DFLL_LOOP_MODE         SYSTEM_CLOCK_DFLL_LOOP_MODE_USB_RECOVERY
#define CONF_CLOCK_DFLL_ON_DEMAND         false

/* DFLL closed loop mode configuration */
#define CONF_CLOCK_DFLL_MAX_COARSE_STEP_SIZE    (0x1f / 8)
#define CONF_CLOCK_DFLL_MAX_FINE_STEP_SIZE      (0xff / 8)
```

In USBCRM mode, some DFLL configuration values are ignored and overridden by the driver as per the USBCRM mode specifications.

The multiply factor (CONF_CLOCK_DFLL_MULTIPLY_FACTOR) is set to 48000 to generate 48MHz from USB SOF reference signal (1kHz). Chill cycles feature (CONF_CLOCK_DFLL_ENABLE_CHILL_CYCLE) is disabled and Quick Lock feature (CONF_CLOCK_DFLL_QUICK_LOCK) is enabled to get a quick lock in the DFLL. Stable tracking after fine lock is disabled since an auto jitter mechanism will be started in USBCRM mode.

2.2.3.2 DFLL Closed Loop Mode

DFLL closed loop mode can be used as a source for both USB device and host operations. But, the reference source should be an external crystal or an external clock input. If any of the internal clock sources is used as reference for DFLL, the output clock will not comply with the USB jitter specifications.

The external crystal/clock should be enabled in conf_clocks.h.

Example Configuration of external crystal/clock as source for GCLK_GENERATOR_2 in conf_clocks.h

```
/* SYSTEM_CLOCK_SOURCE_XOSC32K configuration - External 32 kHz crystal/clock
oscillator */
#define CONF_CLOCK_XOSC32K_ENABLE                  true
#define CONF_CLOCK_XOSC32K_EXTERNAL_CRYSTAL        SYSTEM_CLOCK_EXTERNAL_CRYSTAL
#define CONF_CLOCK_XOSC32K_STARTUP_TIME            SYSTEM_XOSC32K_STARTUP_65536
#define CONF_CLOCK_XOSC32K_AUTO_AMPLITUDE_CONTROL false
#define CONF_CLOCK_XOSC32K_ENABLE_1KHZ_OUPUT       false
#define CONF_CLOCK_XOSC32K_ENABLE_32KHZ_OUTPUT     true
#define CONF_CLOCK_XOSC32K_ON_DEMAND               true
#define CONF_CLOCK_XOSC32K_RUN_IN_STANDBY          true

/* Configure GCLK generator 2 (RTC) */
#define CONF_CLOCK_GCLK_2_ENABLE                 true
#define CONF_CLOCK_GCLK_2_RUN_IN_STANDBY         true
#define CONF_CLOCK_GCLK_2_PRESCALER              1
#define CONF_CLOCK_GCLK_2_OUTPUT_ENABLE          false
```

The source should be one of the external crystal/clock inputs (XOSC32K or XOSC).

```
/* configure external 32kHz Crystal XOSC32K as source */
#define CONF_CLOCK_GCLK_2_CLOCK_SOURCE           SYSTEM_CLOCK_SOURCE_XOSC32K
(or)
/* configure external Crystal XOSC as source */
#define CONF_CLOCK_GCLK_2_CLOCK_SOURCE           SYSTEM_CLOCK_SOURCE_XOSC
```

Example Configuration for DFLL Closed Loop Mode with source as GCLK_GENERATOR_2 in conf_clocks.h

```
/* SYSTEM_CLOCK_SOURCE_DFLL configuration - Digital Frequency Locked Loop */
#define CONF_CLOCK_DFLL_ENABLE                   true
#define CONF_CLOCK_DFLL_LOOP_MODE                SYSTEM_CLOCK_DFLL_LOOP_MODE_CLOSED
#define CONF_CLOCK_DFLL_ON_DEMAND                false

/* DFLL open loop mode configuration */
#define CONF_CLOCK_DFLL_COARSE_VALUE             (0x1f / 4)
#define CONF_CLOCK_DFLL_FINE_VALUE               (0xff / 4)

/* DFLL closed loop mode configuration */
#define CONF_CLOCK_DFLL_SOURCE_GCLK_GENERATOR    GCLK_GENERATOR_2
#define CONF_CLOCK_DFLL_MULTIPLY_FACTOR          (48000000/32768)
#define CONF_CLOCK_DFLL_USBCRM                   false
#define CONF_CLOCK_DFLL_QUICK_LOCK               true
#define CONF_CLOCK_DFLL_TRACK_AFTER_FINE_LOCK    true
#define CONF_CLOCK_DFLL_KEEP_LOCK_ON_WAKEUP      true
#define CONF_CLOCK_DFLL_ENABLE_CHILL_CYCLE       true
#define CONF_CLOCK_DFLL_MAX_COARSE_STEP_SIZE     (0x1f / 8)
#define CONF_CLOCK_DFLL_MAX_FINE_STEP_SIZE       (0xff / 8)
```

2.2.4 DPLL as Clock Source

The DPLL can be used as a clock generator for both USB host and device operations. But, the reference source should be an external crystal or an external clock input. If any of the internal clock sources is used as reference for the DPLL, the output clock will not comply with the USB jitter specifications.

The following options are available for DPLL reference:

- SYSTEM_CLOCK_SOURCE_DPLL_REFERENCE_CLOCK_REF0
  XOSC32K
- SYSTEM_CLOCK_SOURCE_DPLL_REFERENCE_CLOCK_REF1
  XOSC – Reference divider might be required to configure if XOSC > 2MHz

- SYSTEM_CLOCK_SOURCE_DPLL_REFERENCE_CLOCK_REF2
  GCLK_DPLL – GCLK Generator should be configured to generate GCLK_DPLL

Preferably, ~2MHz reference input clock is recommended to get a quick lock on the DPLL.

The external crystal/clock input should be enabled in conf_clocks.h. Refer the Section 2.2.3.2 DFLL Closed Loop Modefor example crystal configuration.

Example Configuration for DPLL with XOSC32K reference in conf_clocks.h

```
/* SYSTEM_CLOCK_SOURCE_DPLL configuration - Digital Phase-Locked Loop */
#define CONF_CLOCK_DPLL_ENABLE                    true
#define CONF_CLOCK_DPLL_ON_DEMAND                 false
#define CONF_CLOCK_DPLL_RUN_IN_STANDBY            true
#define CONF_CLOCK_DPLL_LOCK_BYPASS               false
#define CONF_CLOCK_DPLL_WAKE_UP_FAST              false
#define CONF_CLOCK_DPLL_LOW_POWER_ENABLE          true

#define CONF_CLOCK_DPLL_LOCK_TIME
SYSTEM_CLOCK_SOURCE_DPLL_LOCK_TIME_DEFAULT
#define CONF_CLOCK_DPLL_REFERENCE_CLOCK
SYSTEM_CLOCK_SOURCE_DPLL_REFERENCE_CLOCK_REF0
#define CONF_CLOCK_DPLL_FILTER
SYSTEM_CLOCK_SOURCE_DPLL_FILTER_DEFAULT

#define CONF_CLOCK_DPLL_REFERENCE_FREQUENCY       32768
#define CONF_CLOCK_DPLL_REFERENCE_DIVIDER         1
#define CONF_CLOCK_DPLL_OUTPUT_FREQUENCY          48000000
```

2.3 Low Power Management with SAM D21 USB

2.3.1 Sleep Modes with USB Stack

The Sleepmgr (Sleep Manager) service is used to manage the sleep modes with the USB Stack and the SAM D21 USB driver.

To enter a sleep mode in the application, the following function call should be made,

```
sleepmgr_enter_sleep ();
```

This routine enters the sleep mode currently locked by the sleepmgr.

To lock a sleep mode for the sleepmgr to use,

```
sleepmgr_lock_mode (mode);
```

Table 2-3 and Table 2-4 show the different states of the USB stack and the maximum sleep mode locked for that state.

Table 2-3. Sleep Modes in USB Device Mode

USB Device State	Maximum Sleep Mode Locked
UDD_STATE_OFF	SLEEPMGR_ACTIVE
UDD_STATE_SUSPEND	SLEEPMGR_IDLE_2
UDD_STATE_SUSPEND_LPM	SLEEPMGR_IDLE_1
UDD_STATE_IDLE	SLEEPMGR_IDLE_0

Table 2-4. Sleep Modes in USB Embedded Host Mode

USB Embedded Host State	Sleep Mode
UHD_STATE_OFF	SLEEPMGR_STANDBY
UHD_STATE_WAIT_ID_HOST	SLEEPMGR_IDLE_0
UHD_STATE_NO_VBUS	SLEEPMGR_IDLE_0
UHD_STATE_DISCONNECT	SLEEPMGR_IDLE_0
UHD_STATE_SUSPEND	SLEEPMGR_IDLE_2
UHD_STATE_SUSPEND_LPM	SLEEPMGR_IDLE_2
UHD_STATE_IDLE	SLEEPMGR_IDLE_0

When the application makes a call to the sleepmgr_enter_sleep() routine, the device enters the sleep mode locked by the USB stack (based on current state in USB stack).

If a custom power management is required, then the following configuration should be added in the application.

Modify conf_usb.h - Add a define to disable the sleep management service used in USB Stack

```
#define UDD_NO_SLEEPMGR
```

When using a custom power management, the application has to take care that the wake up time from the sleep mode is within the specification limit of the USB operations. The wake up time is calculated taking into account the time taken by the USB Clock source and the CPU clock source to stabilize and to restart the USB operations by the CPU.

2.3.2 Link Power Management (LPM-L1) Protocol for USB Device

The ASF USB stack supports the LPM-L1 protocol. USB_DEVICE_LPM_SUPPORT should be defined in the conf_usb.h to enable the support for LPM-L1 protocol in the stack.

All the USB classes implemented in the ASF have the required Binary Device Object Store (BOS) descriptor and USB Device Capabilities – USB 2.0 Extension descriptors to notify the LPM-L1 support to the host.

For a custom class or a vendor class using the USB stack, the user has to add the usb_dev_lpm_desc_t descriptor for LPM-L1.

Example to enable LPM Support in USB Device Stack

```
#define USB_DEVICE_LPM_SUPPORT
```

Example to add LPM support in USB Descriptors in custom/vendor class

```
/* Devices supporting BOS Descriptor should have a version >= 0201h */
#define USB_VERSION   USB_V2_1

//! USB Device Qualifier Descriptor
COMPILER_WORD_ALIGNED
UDC_DESC_STORAGE usb_dev_lpm_desc_t udc_device_lpm = {
      .bos.bLength               = sizeof(usb_dev_bos_desc_t),
      .bos.bDescriptorType       = USB_DT_BOS,
      .bos.wTotalLength          = LE16(sizeof(usb_dev_bos_desc_t) +
sizeof(usb_dev_capa_ext_desc_t)),
      .bos.bNumDeviceCaps        = 1,
      .capa_ext.bLength          = sizeof(usb_dev_capa_ext_desc_t),
```

```
        .capa_ext.bDescriptorType  = USB_DT_DEVICE_CAPABILITY,
        .capa_ext.bDevCapabilityType = USB_DC_USB20_EXTENSION,
        .capa_ext.bmAttributes     = ((USB_DC_EXT_LPM | USB_DC_EXT_BESL \
          | USB_DC_EXT_BESL_BASELINE_VALID | USB_DC_EXT_BESL_DEEP_VALID \
          | USB_DC_EXT_BESL_DEEP(BESL_4000_US) |
USB_DC_EXT_BESL_BASELINE(BESL_125_US))),
};
```

The following callbacks are used with LPM mode which should be mapped to the application routines in conf_usb.h.

Table 2-5. LPM-L1 Mode Callbacks from USB Device Stack

Parameters	Description
USB_DEVICE_LPM_SUPPORT	Should be defined in conf_usb.h to enable LPM support
UDC_SUSPEND_LPM_EVENT()	Callback function called on a LPM-L1 Sleep Request from Host
UDC_REMOTEWAKEUP_LPM_ENABLE()	Callback function called to enable Remote Wakeup source for a LPM-L1 Remote Wake-up from Sleep
UDC_REMOTEWAKEUP_LPM_DISABLE()	Callback function called to disable Remote Wakeup source for a LPM-L1 Remote Wake-up from Sleep
void udc_remotewakeup(void)	Function to send the upstream L1 Resume or normal USB Resume to host

The application can enter any sleep mode when the UDC_SUSPEND_LPM_EVENT callback is triggered.

When the USB stack manages the sleep modes through sleepmgr, the sleepmgr_enter_sleep() function should be called to enter the sleep mode (SLEEPMGR_IDLE_1) locked by the USB Stack.

If a custom power management is implemented, the wakeup period from the sleep mode should comply with the BESL parameter specified in the LPM Descriptor.

2.3.3 Link Power Management (LPM-L1) Protocol for USB Host

The ASF USB stack has implemented support for the LPM-L1 protocol. USB_HOST_LPM_SUPPORT should be defined in the conf_usb.h to enable the support for LPM-L1 protocol in the host stack.

Example to enable LPM Support in USB Host Stack

```
#define USB_HOST_LPM_SUPPORT
```

The following functions are for using the LPM mode which can be called from the Application to enter LPM modes.

Table 2-6. LPM-L1 Mode Callbacks from USB Device Stack

Parameters/Functions	Description
USB_HOST_LPM_SUPPORT	Should be defined in conf_usb.h to enable LPM support.
bool uhc_suspend_lpm (bool b_remotewakeup, uint8_t besl)	Function to put the device in L1-Suspend mode. Returns false when LPM is not supported by end device.
void uhc_resume(void)	Function to send a L1-Resume or a normal USB Resume to the device
bool uhc_is_suspend(void)	Function to check whether the host has put the devices into Suspend mode.
UHC_WAKEUP_EVENT	Callback function called when a L1-Resume or an Upstream resume is received from the device or when a L1-Resume or a normal Resume is sent by the host.

The device should have sent the required descriptors for LPM during enumeration. If not, the routine uhc_suspend_lpm will return false and the USB host will not use LPM transactions with that device.

2.4 SAM D21 USB Features Support in the USB driver

Table 2-7. SAM D21 USB Features Support in the USB Driver

Feature	Support in USB driver
Multi-packet transfers	Supported (enabled by default)
Feedback Endpoints	Supported
LPM-L1 Protocol	Supported
Ping-pong mode	Not supported
Virtual Pipes	Not supported

By default, the USB driver uses Multi-packet feature for USB transfers. For creating the feedback endpoint, the user has to configure the endpoints as described in Section 1.3.7 Feedback Endpoint.

3. Atmel USB Stack v2

Refer the following Application Notes for using Atmel® USB Stack v2 with SAM D21 for USB device and host operations. They provide the information on the APIs, callbacks in the USB stack and their usage in application.

Atmel AVR®4900: ASF - USB Device Stack

Atmel AVR4902: ASF - USB Device Composite

Atmel AVR4903: ASF - USB Device HID Mouse Application

Atmel AVR4904: ASF - USB Device HID Keyboard Application

Atmel AVR4905: ASF - USB Device HID Generic

Atmel AVR4907: ASF - USB Device CDC Application

Atmel AVR4920: ASF - USB Device stack - Compliance to the Norm and Performance Figures

Atmel AVR4950: ASF - USB Host Stack

4. References

4.1 Device datasheet

The device datasheet contains the block diagrams of the peripherals and details about implementing firmware for the device. It also contains the electrical specifications and expected characteristics of the device.

Datasheet is available on www.atmel.com in the Documents section of Atmel SAM D21 product page.

4.2 ARM documentation on Cortex-M0+ core

- Cortex®-M0+ Devices Generic User Guide revision r0p1
- Cortex-M0+ Technical Reference Manual revision r0p1

4.3 Atmel Studio

The latest version of Atmel Studio can be downloaded from http://www.atmel.com/tools/atmelstudio.aspx.

5. Revision History

Doc. Rev.	Date	Comments
42261A	03/2014	Initial document release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation
1600 Technology Drive
San Jose, CA 95110
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan G.K.
16F Shin-Osaki Kangyo Building
1-6-4 Osaki, Shinagawa-ku
Tokyo 141-0032
JAPAN
Tel: (+81)(3) 6417-0300
Fax: (+81)(3) 6417-0370