USB MAX3421 Host Controller User's Guide

Version 1.00

For use with USBH MAX3421 Host Controller versions 1.03 and above

Date: 05-Mar-2015 15:13

Table of Contents

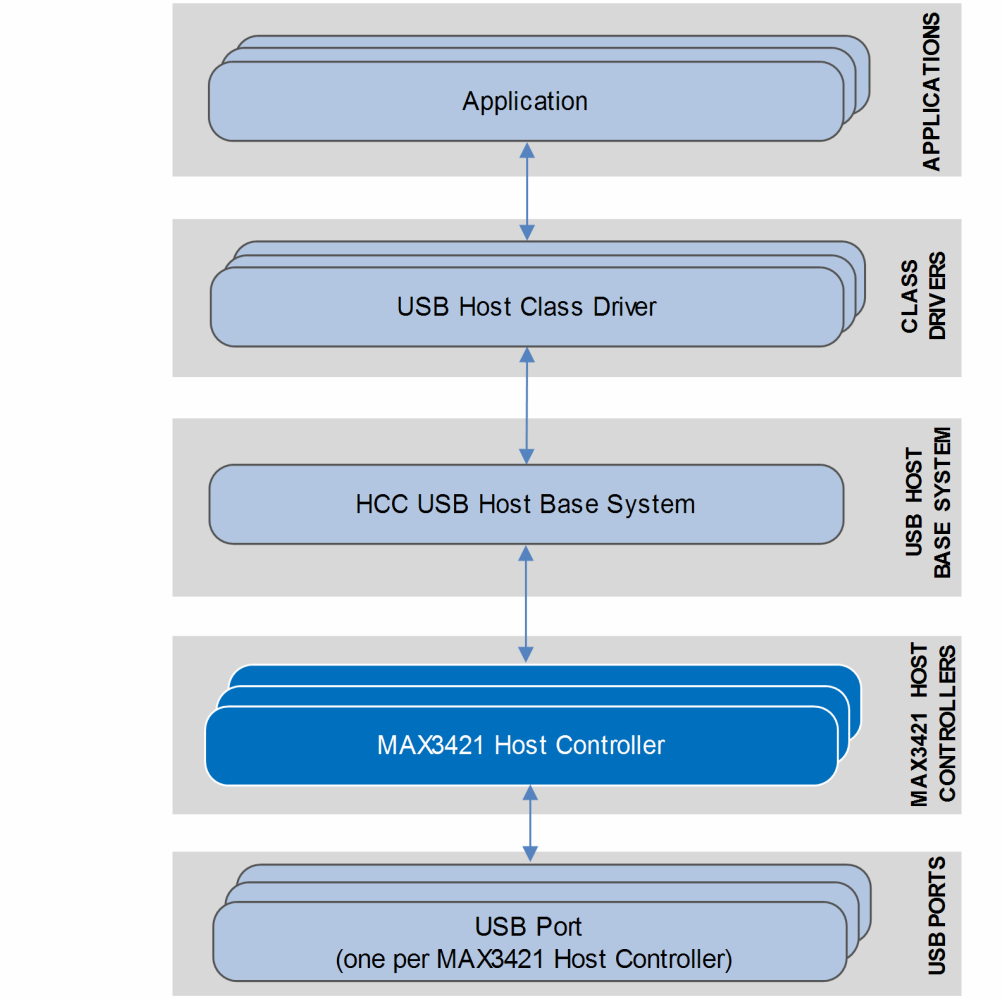
1 System Overview

1.1 Introduction

This guide is for those who want to implement HCC Embedded's MAX2341 USB host controller with the HCC USB host stack.

The MAX2341 module provides a high speed USB 2.0 host controller with both full and low speed USB functions. The controller handles all USB transfer types and, in conjunction with the USB host stack, can be used with any USB class driver. The package is for use with MAX2341 MCUs from Maxim Integrated™.

The position of the host controller within the USB stack is shown below:



This host controller can operate in two modes:

- Traditional mode – there is one physical ISR that sends an event to the transfer task and the transfer task processes the transfers.

- ISR is executed from the task context – this can be required in environments where the physical ISR only sends an event to a task, indicating the occurrence of an IRQ because no peripheral access is allowed from the ISR and it has to complete as fast as possible.

1.2 Feature Check

The main features of the host controller are the following:

- It conforms to the HCC Advanced Embedded Framework.
- It can be used with or without an RTOS.
- It is integrated with the HCC USB Host stack and all its class drivers.
- It can be used with any MAX2341-compliant USB host controller.
- It supports multiple simultaneous MAX2341 controllers, each with multiple devices attached.
- It supports all USB transfer types: Control, Bulk, Interrupt and Isochronous.

1.3 Packages and Documents

Packages

The table below lists the packages that you need in order to use this module:

Package	Description
hcc_base_doc	This contains the two guides that will help you get started.
usbh_base	The USB host base package. This is the framework used by USB class drivers to communicate over USB using a specific USB host controller package.
usbh_drv_max3421	The USB MAX3421 host controller package described by this document.

Documents

Readers should note the points in the HCC Documentation Guidelines on the HCC documentation website.

HCC Firmware Quick Start Guide

This document describes how to install packages provided by HCC in the target development environment. Also follow the *Quick Start Guide* when HCC provides package updates.

HCC Source Tree Guide

This document describes the HCC source tree. It gives an overview of the system to make clear the logic behind its organization.

HCC USB Host Base System User's Guide

This document defines the USB host base system upon which the complete USB stack is built.

HCC USB MAX3421 Host Controller User's Guide

This is this document.

2 Source File List

This section describes all the source code files included in the system. These files follow the HCC Embedded standard source tree system, described in the *HCC Source Tree Guide*. All references to file pathnames refer to locations within this standard source tree, not within the package you initially receive.

> **Note:** Do not modify any of these files except the configuration file and PSP files.

2.1 API Header File

The file **src/api/api_usbh_max3421.h** is the only file that should be included by an application using this module. It declares the API functions.

2.2 Configuration File

The file **src/config/config_usbh_max3421.h** contains all the configurable parameters. Configure these as required. For details of these options, see Configuration Options.

2.3 Source Code Files

The source code files are in the directory **src/usb-host/usb-driver/max3421**. **These files should only be modified by HCC.**

File	Description
max3421_drv.c	Register access functions.
max3421_drv.h	Header file for register access functions.
max3421_reg.h	MAX3421 register file.
max3421_uh.c	Main code file.
max3421_uh.h	Header file for MAX3421 public functions.
max3421_uh_hc.c	HC descriptor structures.
max3421_uh_hc.h	HC MAX3421 specific header file.
max3421_uh_hub.c	Main hub code.
max3421_uh_hub.h	Public MAX3421 hub functions.

2.4 PSP Files

These files are in the directory **src/psp/target/usb-host-max3421**. They provide functions and elements the core code may need to use, depending on the hardware.

Note: These are PSP implementations for the specific microcontroller and development board; you may need to modify these to work with a different microcontroller and/or board. See Hardware-Specific Functions for details.

File	Description
max3421_uh_hw.c	Functions source code.
max3421_uh_hw.h	Functions header file.

2.5 Version File

The file **src/version/ver_usbh_max3421.h** contains the version number of this module. This version number is checked by all modules that use this module to ensure system consistency over upgrades.

3 Configuration Options

Set the system configuration options in the file **src/config/config_usbh_max3421.h**. This section lists the available configuration options and their default values.

MAX3421_UH_TRANSFER_TASK_STACK_SIZE

The stack size of the MAX3421 transfer task(s). The default is 1024.

This option is ignored if MAX3421_UH_ISR_TASK is enabled because in that case the stack size is provided by the task the ISR function is called from.

MAX3421_UH_ISR_TASK

The default is zero. Set this to 1 if either of the following applies:

- The ISR is executed from a task context. In this case the transfer task is not created but is directly called from the interrupt.
- The system is used without interrupts in non-OS mode.

MAX3421_MAX_EP

The maximum number of Bulk and Interrupt endpoints. The default is 5.

MAX3421_ONE_NAK_PER_FRAME

Set this to 1 (the default) to allow only one NAK for an endpoint in a frame. This can be useful to decrease interrupt load as this chip generates an interrupt for every single NAK.

In order to minimize the USB interrupt load, consider the following:

1. If NO RTOS is used, SOF interrupts are needed for timeout handling. Therefore they are always enabled, regardless of whether there is an active transfer on an endpoint which needs them.
2. If an RTOS is used, the following are valid:

- A SOF interrupt fires every millisecond in case there is at least one active transfer which needs it. Such transfers are all Interrupt transfers and Bulk transfers if MAX3421_ONE_NAK_PER_FRAME is set to a non-zero value.
- When MAX3421_ONE_NAK_PER_FRAME is set to zero, data sent to or requested from devices which answer slowly may generate NAK bursts on the bus, every one of which generates an interrupt. During idle periods, however, no interrupt is generated at all.
- When MAX3421_ONE_NAK_PER_FRAME is set to a non-zero value, no NAK bursts occur so no interrupt bursts will appear. Slow Bulk responses are polled every millisecond instead. As in the previous case, no SOF interrupts will appear during idle periods.

In summary, for systems using Bulk transfers only (like pure Mass Storage communication), setting MAX3421_ONE_NAK_PER_FRAME to non-zero results in lower interrupt load but possibly slower communication.

MAX3421_UH_ISR_ID

The ISR ID used when the ISR is installed. The default is zero.

MAX3421_UH_INT_PRIO

The ISR priority used when the ISR is installed. The default is zero.

MAX3421_LEVEL_INTERRUPT

Specify whether the generated interrupt should be edge (zero, the default) or level (1) active.

MAX3421_EDGE_POLARITY

If MAX3421_LEVEL_INTERRUPT is edge-active, the polarity of the interrupt: negative (zero, the default) or positive (1).

MAX3421_PULSE_WIDTH

If MAX3421_LEVEL_INTERRUPT is edge-active this option defines the pulse width of the interrupt. Possible values are:

Value	Pulse width
0	10.6µs (the default)
1	5.3µs
2	2.6µs
3	1.3µs

MAX3421_UH_SPI_SINGLE_MODE

This controls whether MAX3421 is the only slave unit on the SPI bus. The options are:

- Zero – there are multiple slave units on the SPI bus. This is the default.
- 1 – MAX3421 is the only slave unit.

MAX3421_UH_SPI_UID

The SPI unit ID. The default is zero.

4 Starting the MAX3421 Controller

This section shows how to start the host controller and describes the task created. It also describes the pin functions. It includes a code example.

4.1 API Functions

This section documents the Application Programming Interface. It includes all the functions that are available to an application program.

usbh_max3421_hc

This external interface function provides the host controller descriptor required by the **usbh_hc_init()** function.

Format

```
extern void * const  usbh_max3421_hc
```

max3421_get_iopins

Use this function to get the state of the general input and output pins of the MAX3421 chip, the IOPIN1 and IOPIN2 registers.

Format

```
int max3421_get_iopins (
    uint8_t *   p_ipins,
    uint8_t *   p_opins )
```

Arguments

Parameter	Description	Type
p_ipins	Where to write the state of the input pins.	uint8_t *
p_opins	Where to write the state of the output pins.	uint8_t *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

max3421_set_opins

Use this function to set the state of the output pins.

Format

```
int max3421_set_opins ( uint8_t opins )
```

Arguments

Parameter	Description	Type
opins	The output pin state values.	uint8_t

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

4.2 Host Controller Task and ISR

Task

There are two options, depending on the setting of MAX3421_UH_ISR_TASK:

- If MAX3421_UH_ISR_TASK is not set, the host controller task handles all completed transfers. Callback requested for the transfer is executed from this task. The task has the following attributes:

Attribute	Description
Entry point	*usbh_max3421_transfer_task*
Priority	USBH_TRANSFER_TASK_PRIORITY
Stack size	MAX3421_UH_TRANSFER_TASK_STACK_SIZE. The default is 1024.

- If MAX3421_UH_ISR_TASK is set, the transfer task is executed from the ISR function. This is executed from an external task context and the priority and stack size must be set for the task the ISR function is called from.

ISR

There is one ISR if MAX3421_UH_ISR_TASK is not set, none if it is set.

4.3 Code Example

This example shows how to initialize the host controller. Note the following:

- There is only one external interface function, **usbh_max3421_hc()**. To link this host controller to the system, you call the **usbh_hc_init()** function with this function as a parameter.
- The last parameter in the **usbh_hc_init()** call is the number of the host controller.

```
void start_usb_host_stack ( void )
{
int  rc;
rc = hcc_mem_init();
    if ( rc == 0 )
    {
        rc = usbh_init();  /* Initialize USB host stack */
    }
    if ( rc == 0 )
    {
        /* Attach MAX3421 host controller */
        rc = usbh_hc_init( 0, usbh_max3421_hc, 0 );
    }

    if ( rc == 0 )
    {
        rc = usbh_start();  /* Start USB host stack */
    }
    if ( rc == 0 )
    {
        rc = usbh_hc_start( 0 );  /* Start MAX3421 host controller */
    }
.....
}
```

5 Integration

This section specifies the elements of this package that need porting, depending on the target environment.

5.1 OS Abstraction Layer (OAL)

All HCC modules use the OS Abstraction Layer (OAL) that allows the module to run seamlessly with a wide variety of RTOSes, or without an RTOS.

This module requires the following OAL elements:

OAL Resource	Number Required
Tasks	1
Mutexes	1
Events	1
ISRs	1

5.2 PSP Porting

The Platform Support Package (PSP) is designed to hold all platform-specific functionality, either because it relies on specific features of a target system, or because this provides the most efficient or flexible solution for the developer. For full details of any function, see the relevant *PSP User Guide*.

The module makes use of the following standard PSP functions:

Function	Package	Element	Description
psp_memset()	psp_base	psp_string	Sets the specified area of memory to the defined value.
psp_spi_cs_lo()	psp_base	psp_spi	Sets chip select low.
psp_spi_cs_hi()	psp_base	psp_spi	Sets chip select high.
psp_spi_lock()	psp_base	psp_spi	Locks the SPI for the specific unit. This may be useful if multiple units are attached to the same SPI bus.
psp_spi_unlock()	psp_base	psp_spi	Unlocks the SPI for the specific unit. This may be useful if multiple units are attached to the same SPI bus.

Hardware-Specific Functions

These functions are provided by the Platform Support Package to perform various tasks. They are designed for a specific microcontroller and development board. You may need to port them to work with your hardware solution; they are designed to make porting easy.

The package includes samples in the file **max3421_uh_hw.c**.

max3421_uh_hw_init

This function is provided by the PSP to initialize the device.

Format

```
int max3421_uh_hw_init ( t_usbh_unit_id unit )
```

Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

max3421_uh_hw_start

This function is provided by the PSP to start the device.

Format

```
int max3421_uh_hw_start ( t_usbh_unit_id unit )
```

Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

max3421_uh_hw_stop

This function is provided by the PSP to stop the device.

Format

```
int max3421_uh_hw_stop ( t_usbh_unit_id unit )
```

Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

max3421_uh_hw_delete

This function is provided by the PSP to delete the device, releasing associated resources.

Format

```
int max3421_uh_hw_delete ( t_usbh_unit_id unit )
```

Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.