USB LLD Readme File

Microcontrollers

Never stop thinking.

Edition v4.2, 24 Jan 2006

Published by Infineon Technologies India pvt. Ltd.,
ITPL,
Bangalore, INDIA

Attention please!

The information herein is given to describe certain components and shall not be considered as warranted characteristics.

Terms of delivery and rights to technical change reserved.

We hereby disclaim any and all warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

Infineon Technologies is an approved CECC manufacturer.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office in Germany or our Infineon Technologies Representatives worldwide (see address list).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

USB LLD Readme File

Microcontrollers

Never stop thinking.

Table 1

Page	Subjects (major changes since last revision)
ALL	First Draft

Author:
Jayashree Badarinath

We Listen to Your Comments

Is there any information in this document that you feel is wrong, unclear or missing ? Your feedback will help us to continuously improve the quality of our documentation. Please send your feedback (including a reference to this document) to:

care_services@infineon.com

1 Readme for USB LLD

1.1 Tools Used with USB LLD

Tasking 2.2R3

GNU 3.3.7.3

PLS UAD 1.9.12

Hardware : TC1130 - A Step board

TC1130 - BA Step Board

TC1130 - BB Step board (Latest)

(The package has been tested with both manual and automatic mode(Automatic mode works only with TC1130 BB Step board) Refer to Demo application foder to get started with the USB LLD.

1.2 Hardware Restrictions

1) 2 kBytes buffer size (User must take care not to exceed this bandwidth).
2) 1 Configurations (CF1)

CF1 with 4 interfaces

IF0 with 1 Alternate Setting AS0

IF1 with 2 Alternate Setting AS0 and AS1

IF2 with 3 Alternate Settings AS0, AS1 and AS2

IF3 with 3 Alternate SettingsAS0, AS1 and AS2

Note: The programmer may associate any Endpoints with any existing Interface and Alternate Setting as long as this meets the above said requirements. For further reference please refer to USB Section of TC1130_UMPU_V10D2.

1.3 Limitations:

1. **Tasking compiler** has MMU alignment from version 2.2r2 onwards, for which patches have been provided with the release package

2. **Short packet transmission**
2.1 Short packet transfer does not work with BSZ > 1 (For manual mode)

2.2 **ISO short packet transmission** for IN packet does not work, short packets need to be filled with zero until USB Packet size.

In Automatic mode

2.3 Short Packets from Host to Device will be filled with zeros until USB packet size.
A full USB Packet followed by a packet smaller than 9 bytes causes an error in USB Device if both packets are within one USB Frame. This is only valid for direction Host to Device.

3. **Stall Endpoint**
Stall Endpoint function does not work proper.

4. **Host controller - Enhanced mode**:
4.1 With the enhanced host controller, enumeration fails with Internal RAM memory in combination with 144MHz frequency.
4.2 With enhanced host controller, tranmission fails after several transfers for configuration of external memory with 144MHz.

1.4 C source files

File name	Description
usb_idl.c	Implements low-level driver functionalities for USB device.
usb_iil_setup.c	Implements device enumeration functionalities.
usb_iil_rx.c	Implements reception functionality for application layer
usb_main_das.c	*Implements the sample program to test DAS Application*

1.5 Header Files

File name	Description
usb_iil_cfg.h	Configuration file for user as per USB1.1/1.0 specs.
usb_idl_cfg.h	USB hardware configurations file for user.
usbd_idl_macro.h	Abstracts hardware definitions for low-level driver.
usbd_idl.h	Implements the hardware definitions.
usb_iil_setup.h	Contains structure definitions for USB1.1 specs parameters.
usb_iil_common.h	Contains defines which are common to all USB HAL files
usb_iil_api.h	Contains interface definitions for application layer.

Other dependent files from system folder

File name	Description
compiler.h	Contains complier dependent definitions
sys_api.h	Contains system related definitions
sys_cfg.h	System configuration file for user for setting interrupt priorities.

sys_iil.c	Contains system related functions
common.h	Contains system related functions
main.h	just main header file

Files to be used for compilation

==========

1. USB

The required files are

The following files are in the **Src** directory

usb_iil_cfg.h

usb_iil_api.h

usb_idl_cfg.h

usb_iil_common.h

usb_iil_setup.h

usbd_idl.h

usbd_idl_macro.h

usb_idl.c

usb_iil_rx.c

usb_iil_setup.c

The following files are in the **System** directory

common.h,

compiler.h,

sys_api.h,

sys_cfg.h,

sys_iil.c

main.h

usb_main_das.c-- This file is used to test USB driver.

/* @def USBD_MANUAL_MODE | USBD_AUTO_MODE This defintion is for switching between the automatic and manual mode. USBD_MANUAL_MODE for manual mode and USBD_AUTO_MODE for automatic mode */

#define **USBD_MANUAL_MODE**
//#define **USBD_AUTO_MODE**

Few Testing guidelines

==========

For testing with
EP1 and EP2: IF1ASI
EP3 and EP4 IF2AS1
EP5 and EP6 IF2AS2
EP7 and EP8 IF3ASI
EP9 and EP10 IF3AS2
#define CONFIG_USB_TOTBYTES 0xA1instead of 0xA0 in USB_IIL_CFG.H file.
due to ZLP Hardware BUG

==========

To Test on Flash memory using Flashload

==========

1. Include the Crt0.S and EBUSettings.S file from the GNU\Flash along with the required *.c and *.h files.
2. Copy the target.ld to the working directory.
3. In Link Options give -nostartfiles and Link Script as target.ld
4. Start the flash-jtagsrv.bat from the FlashLoad directory provided with the starter kit CD (Note: No other Jtag server should be running)
5. copy the USB LLD compiled elf file to obj directory of the Flashload directory.
6. Download the Flashload.elf from Flashload(from Starter kit CD) /obj directtory with Virtual I/O window open using GNU Debugger.
7. Run the image. and type *.elf in the Virtual I/O, wait until the successful write of the image on to flash.

8. Change the Jumper setting to off-off-on and reset the board, Plug out and plug in the cable for transmission of data.

[For further reference to the Flashloader refer to TC1130 Triboard - Getting Started - v1.3.ppt]

==========

==========

EP1 and EP2
BSZ == 1 to 8
Bulk mode of transmission Passed Ok for multi-ple of 64 Bytes
(Note: This fails when executed from flash if it is non multiple of 64 Bytes; Explanation: This depeneds on the execution of the interrupt routine which is really controlling the corruption of the FIFO pointers. In case of short packets, you will receive the second packet very fast and by the time you come and make the fifo as invalid, the hardware pointers would have got corrupted. Since when you execute the code from flash, interrupt execution gets slower.)

==========

DASCFI Testbench

==========

Hidden string needs to be changed for the following in the usb_iil_cfg.h
Passed Ok

```
static USB_HIDDEN_STRING_DESCRIPTOR
USB_Hidden_string_descriptor =
{
   sizeof(USB_Hidden_string_descriptor),     /* @field bLength */
   USB_STRING_DESCRIPTOR_TYPE,               /* @field bDescrip-
torType */
   {                                 /* TC1130A USB*/
   'D', 0,
   'E', 0,
   'V', 0,
   'I', 0,
   'C', 0,
   'E', 0,
```

```
    'O', 0,
    'N', 0,
    'E', 0,
    '1', 0,
    '1', 0
    }
};
```

2 Application Hints

2.1 USB_LLD.H1 Padding of Short Packets

In order to achieve maximum performance with TC1130 USB module the number of IN buffers (Host receives) is set to 6. Due to a hardware issue, Short Packets (Data size less than USB packet size), which are sent from application to Host, have to be filled up with zeros until packet is full. If there is only one buffer for IN Endpoint, the padding of zeros is not necessary. For OUT (Host sends) direction Endpoints the use of Short Packets is valid. No padding of zeros is necessary.
The code example (Packet size = 64bytes) shown below is used in the file usb_main_das.c and can be used in a user application.

```
if ((expected_das_pkt_length%64) != 0)
{
// save odd length
old_expected_das_pkt_length = expected_das_pkt_length;
// calculate new expected_das_pkt_length
expected_das_pkt_length = expected_das_pkt_length -
(expected_das_pkt_length % 64) + 64;
for (i = old_expected_das_pkt_length; i < expected_das_pkt_length; i++)
{
main_buffer[i] = 0;
}
}
```

Automatic mode Application hints

If the user wants to use the automatic mode in order to reach the maximum performance, he has to take care about the following points.

1. USB Driver Buffer on Host

In order to reveive more than one USB packet per USB Frame, the buffer of the driver has to be initialized with a size which is a multiple of USB packet size e.g. 2048

Usually, the application on Host will collect USB packets from this buffer when it is full. Due to the big buffer, some packets could remain in buffer and would never be collected, because the USB is done with transfer.

To avoid this, the USB provides two functions which should be used in automatic mode. These functions will be described in the following.

2. Function: **USBD_SET_DONE_BIT()**

If the firmware application reached the last packet to transfer to host, it has to check the length of this packet. If this packet is shorter than USB packet size, user has to call the function USBD_SET_DONE_BIT(Endpoint) in order to signal the end of transfer to the host and to force the collecting of packets out of buffer

3. Function: **USBD_TX_ZERO_LENGTH_PKT()**

If the length of the last packet is equal to USB packet size, user has to call the function USBD_TX_ZERO_LENGTH_PKT(Endpoint) in order to signal the end of transfer to the host and to force the collecting of packets out of buffer.

Note:

1. Function USBD_TX_ZERO_LENGTH_PKT() should not be used, if buffer size on Host is equal to USB Packet size.
2. It is not allowed to use both functions after the last transfer.

3 Project workspace

GNU

Copy the above mentioned files to the desired directory:

1. Common.h, compiler.h, sys_api.h, sys_cfg.h and sys_iil.c from system folder
2. Usb_idl.c, usb_iil_rx.c, usb_iil_setup.c, usb_idl_cfg.h, usb_iil_api.h, usb_iil_cfg.h, usb_iil_common.h, usb_iil_setup.h, usbd_idl.h, usbd_idl_macro.h, usb_main_das.c [Version 4.2]

Note: product ID and BCD Device need to be **0x1F** CONFIG_USB_DEVICE_DESCRIPTOR in usb_iil_cfg.h [Note: This is required for USBIO Light driver]

```
0x001F,  /* field ProductId assigned by Infineon */\
0x001F,  /* field bcdDevice */\
```

4 Copy the target.ld from USB_V4.2\GNU\GNU_Target_Files\GNU_3.3.7.3.

And make the workspace with the following project settings:

Build Rules:

User Flags: -mall-errata
Standard Debug
Level2
Tv1.3

Defines:
TRIBOARD_TC1130

LINK RULES:

o/p: usb_lnk.elf

Link Flags: -mtc13 -Wl,-Map -Wl,usb_internal.lst

Link Script
//PATH/target.ld

Build the .elf.

Tasking

Copy the following files to the desired directory:
1. Common.h, compiler.h, sys_api.h, sys_cfg.h and sys_iil.c from system folder
2. Usb_idl.c, usb_iil_rx.c, usb_iil_setup.c, usb_idl_cfg.h, usb_iil_api.h, usb_iil_cfg.h, usb_iil_common.h, usb_iil_setup.h, usbd_idl.h, usbd_idl_macro.h, usb_main_das.c [Version 4.2]

Apply Tasking Patches
1. In addition copy the cstart.asm, nommu.lsl and nommu.opt, from USB_V4.2\Tasking_Patches to the current project workspace directory
2. Load the nommu.opt, from the project folder which was copied earlier.
3. Change the path to the nommu.lsl to the current project directory.
4. Ensure the cstart.asm copied into project folder is read-only so that the changes are not overwritten.

> The USB LLD is working fine with Tasking 2.2r2 and Tasking 2.2r3 with the MMU library off with the following workaround defined below.
> >>>>>
> **MMU libraries**
> Special MMU libraries have been added for derivatives that have a

MMU on board. These libraries can be found in the subdirectory lib/tc1_mmu/ and lib/tc2_mmu. The MMU hardware workaround is triggered by the --mmu-present option. This option is automatically set when targets are specified (with the -C option) which have a MMU. So -Ctc1130 also sets the --mmu-present option, removing the -Ctc1130 is the only option to check this, but this has some side effects. These MMU libraries contain a natural alignment for data objects. These additional MMU libraries are required, as the MMU requires natural alignment. This is causing alignment issues in the low level driver code.
Concerning this MMU matter, please have a look at the errata sheet of the TC1130, silicon bug CPU_TC052. Here the problem is described and further information is given.

When this library is enabled, the MMU library alignment is overwriting the data alignment in the low level driver code due to which the transmission and reception of data is failing

[Workaround] - Select user CPU type EDE will use '-Cuserdef113' to identify a user CPU type. Its corresponding register file only contains the default CPU registers. The assembler still requires the register definitions for tc1130 which can be explicitly added to the 'include this file before source' EDITBOX of the assembler 'preprocessing' PAGE. In the Tasking_Patches folder you will find an option file (nommu.opt) which can be imported into project to make this happen. This option file will also setup your 'script file' PAGE such that it uses external LSL file nommu.lsl, and just a definition additon in cstart.asm will make it work.

Even if the MMU library is enabled with this option file code would still work fine(Hence MMU library and USB can still be integrated in case if required), and hence this workaround does not have any side effects, only thing is we need use the .opt file, .lsl and cstart.asm for Tasking

2.2r2 onwards. User need to make sure to include the correct def files, as well as the correct LSL file for the linker. These files are all automatically included by the -Ctc1130 option (you can verify this by using the -v option for the control program, this allows you to see which options are passed to the tools).

Compile the code

Refer to Demo_Application\Quick_LLD_Reference directory to get started with the USB LLD

3.1 TC1130 USBIO Light Driver Installation

Uninstall the existing inf files using
TC1130_LLD_V4.00\USB_V4.2\USB_Driver\USBIOcw.EXE

Install the driver usbiov.inf present in driver folder.
[Note: This works only for 12 hours, and after that PC would require reboot.

http://www.infineon.com

Published by Infineon Technologies AG