USB HAL API User Guide

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 HAL API User Guide

Microcontrollers

Never stop thinking.

USB **Revision History 24 Jan 2006** v4.2	
Draft Version 0.0.1	
Page	Subjects(major changes since last version)
All	Draft version created
All	Updated Appendix B and section 5. Added Appendix C, section 4, and section 6. Version number changed to 2.0 for the Implementation Release SW_M7.
All	Version 2.5 :- Added chapter for eLinux and Added Chapter12 (Limitations for USB LLD Release 2.5)
All	Version 2.6 :- Updated Release Notes, Limitations, eLinux and quick start sections. Changes to meet the USB LLD Release 2.6
All	Version 2.7 Hidden string support from release 2.6 Removed. Release notes updated.
All	Version 3.0 Padding of zeros support from release 2.7 Removed. Release notes updated. Additional chapter for user guide added.
All	Version 4.2 Additional macro and limitation chapter has been updated

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 USB HAL

1.1 Introduction

The USB HAL (hardware abstraction layer) forms one part out of a larger system of software modules designed to buffer application software from hardware specific implementation details. The Infineon Technologies modular HAL approach however goes beyond simply providing a standardised API for a type of device, each HAL is designed to work as part of a larger system. System resources are reserved by the HAL's using a system hardware abstraction layer meaning that runtime conflicts are avoided and different peripherals may use the same resources at different points in the application.

In cases where these features are not desirable, possibly due to runtime efficiency or code size constraints, they can simply be removed by modifying a single configuration file and recompiling the HAL, meaning there is no unnecessary code overhead.

In summary the modular HAL system provides the following advantages:

–No extra hardware specific code is required

–Pre-tested code modules are available

–Hardware can be exchanged with little or no application software modifications

–Power saving features incorporated in the HAL

–System resource conflicts are automatically avoided

–Data transfer requests can be queued.

–HAL's are highly configurable

–Porting to different hardware is easy

–Standardised API can be used for development

–Application and hardware dependant developments can go in parallel assuming a standard API

1.2 USB HAL 4.2 Release Notes

The HAL release verison 4.2 supports following features:

-FULL Speed mode support(12 Mbps)

-USB 1.1 Device Framework

-Uses internal transeceiver

-Synchronous data transfer for reception and transmission.

-Endpoint Interrupts are supported on different fixed interrupt nodes to reduce the load on CPU.

-Internal or External USB clock support.

-Supports embedded linux for TC1130.

-Supports the device enumertion process and is transparent to user

-Supports static configuration of parameters according to USB1.1 specification.

- Hardware FIFO size is configurable (Maximum size * Number of Blocks) for each end point. This feature helps to reduce the load on CPU i.e, for reception, data is buffered to be retrived by the CPU at a later stage. For transmission, data is simply put into FIFO for later access by UDC.

- FIFO Handling is done via manual method.

- Supports for automatic FIFO Mechanism with only TC1130 BB Step board.

- Suppots all four data transfer types (Control, ISO, Interrupt and Bulk)

Changes in this release compared to previous release v3.0

Please see the Appendix E for more details.

2 Quick start for USB HAL

2.1 USB LLD with released package

The USB released package comes with ready to use preconfigured endpoints and interfaces. The user needs to enable the SYS_CFG_USB_DEVICE_ENABLE macro in the **sys_cfg.h** before compiling and test the USB functionality. Once the image is running on the target, the user needs to plug and play the USB cable so that device is recongnized and enumerated by the host system(e.g, Windows). Then user can proceed to test the application or build his own applications. User can also configure for the system clock (48Mhz, 96Mhz or 144Mhz) using the parameter SYS_CLOCK_96MHZ in sys_cfg.h which accordingly sets the N, P, K values for the desired frequency. **To change the automatic mode and manual mode of working, the macro USBD_MANUAL_MODE or USBD_AUTO_MODE needs to be set in usb_idl_cfg.h. (Automatic mode works only with the TC1130 BB Step board).**

2.2 USB LLD with DAvE

- The user needs to configure the system clock(48Mhz, 96Mhz, or 144Mhz)
- Configure the SCU register to derive USB Clock (48Mhz) from system clock ratio.
- Enable the global interrupt.
- Open the USBD device configuration. Enable the device. Choose the default settings.
- Generate the code and test it.

The user needs to set the application interfaces as defined in **usb_iil_api.h** (also see the application example section) in this document) to test the applications.

2.3 Other Pointers

If the user wishes to configure the interfaces, alternate settings and other pipe properties to meet their requirement, then user has to configure the following three files.

1. **usb_iil_cfg.h**
   - Configure the USB 1.1 Specification device framework parameters.
2. **usb_idl_cfg.h**
   - Configure the TC1130 USB hardware configurations.

The configuration of parameters in the above two files are explained in the HAL Configuration section in this document.

Please refer to APPENDIX D for list of files that needs to be compiled

3 USB HAL Application Program Interface

This section defines the interface between the peripheral hardware abstraction layer and the application software. It details the constants, typedef names, function names and limitations of the HAL.

3.1 USB_API_V_MAJ macro

```
#define USB_API_V_MAJ
```

Defined in: USB_IIL_API.H

API Major Version Number, USB_API_V_MAJ is defined to the major version of this API which the USB HAL supports. This version is defined as 0.1. Application software may check this field to determine if the HAL API version is acceptable.

3.2 USB_API_V_MIN macro

```
#define USB_API_V_MIN
```

Defined in: USB_IIL_API.H

API Minor Version Number, USB_API_V_MIN is defined to the minor version of this API which the USB HAL supports. This version is defined as 0.1. Application software may check this field to determine if the HAL API version is acceptable.

3.3 USB_BUFFER_CB Structure

```
typedef struct {
IFX_UINT32 ep_number;
IFX_UINT32 cb_number;
IFX_UINT32 bytesRequested;
IFX_UINT32 bytesReceived;
IFX_UINT32 status;
void * pBuffer;
      void (*USB_application_call_back) (void *pCB);
} USB_BUFFER_CB;
```

usb buffer control block for reception and transmission. The handling of this control block and its synchronization is taken care at the USB_receive interface level.

Defined in: **usb_iil_api.h**

3.3.1 Members

ep_number
endpoint number for this communication

cb_number
control block number. Not to be used

bytesRequested
Number of bytes requested (reception/transmission) for this buffer. To be filled by application layer.

bytesReceived
Number of bytes received/transmitted for this buffer .Filled by low-level driver.

status
status of this control block. See USB_CB_BUFFER_STATUS

pBuffer
pointer to application supplied buffer

void (*USB_application_call_back) (void *pCB)
Application call back funtion for low-level driver. NULL if none. CallBack not supported.

3.3.1.1 Comments

In current release used only for reception.

3.4 USB_CB_BUFFER_STATUS

```
enum USB_CB_BUFFER_STATUS {
CB_STATUS_FREE,
RX_STATUS_PENDING,
RX_STATUS_FILLED,
TX_STATUS_PENDING
};
```

Holds the receive control block status.
Defined in: **usb_iil_api.h**

3.4.1 Members

CB_STATUS_FREE
Rx buffer is free. Filled by low-level driver

RX_STATUS_PENDING
Read pending. To be filled by appilcation layer.

RX_STATUS_FILLED
Read complete. Not used

TX_STATUS_PENDING
transfer pending. Not used

3.5 USB_NUMBER_OF_CB macro

```
#define USB_NUMBER_OF_CB
```

Defined in: **usb_iil_api.h**

Number of USB_BUFFER_CB control blocks used by application. Currently set to 2. The user can initialize based on number of simultaneous request being made at receive interface.

3.6 USB_DEVICE_STATES

```
enum USB_DEVICE_STATES {
USB_IDLE,
USB_ATTACHED,
USB_RESET,
USB_CONFIGURED,
USB_SUSPENDED,
};
```

The USB_DEVICE_STATES holds various usb device enumeration and control states.

Defined in: **usb_iil_api.h**

3.6.1 Members

USB_IDLE
USB device is not connected.

USB_ATTACHED
USB device connected.

USB_RESET
USB device has been reset by host & a unique address assigned

USB_CONFIGURED
USB device has been configured by host and host can now use device functions (transmission or reception on endpoints can begin).

USB_SUSPENDED
USB device is in suspende mode

3.7 USB_STATUS

```
enum USB_STATUS {
USB_OK,
USB_ERROR
};
```

The USB_STATUS holds various usb error constants returned by funtions being called by appilcations. This is the only error mechanism avaiable for appilcation interface. So appilcation callee should check, specifically for USB_ERROR, and proceed further.

Defined in: **usb_iil_api.h**

3.7.1 Members

USB_OK
Funtion call successful

USB_ERROR
Funtion call unsucessful

3.8 USBD_device_initialize

USB_STATUS USBD_device_initialize(void)

USB device initialization function. This function initializes all the usb resources and the usb device itself. The usb and system clocks should be initialized before calling this routine.

Defined in: **usb_iil_api.h**

3.8.1 Comments

After initialization the application needs to check for USB_CONFIGURED device state for configuration before transmitting or reception on end point. The configuration files **usb_iil_cfg.h** and **usb_idl_cfg.h** needs to be initialized for proper device funtioning.

Returns USB_OK if device initialization is success else USB_ERROR

3.9 USBD_get_device_state

IFX_UINT8 USBD_get_device_state(void)

USB device state. This function returns the USB_DEVICE_STATES defined above. The application can check for device conditions like attached, idle etc. After device initialization the application needs to check for USB_CONFIGURED device state for configuration before transmitting or reception on end point.

Returns the current device state.

Defined in: **usb_iil_api.h**

3.10 USBD_transmit_ep0

IFX_SINT32 USBD_transmit_ep0(IFX_UINT8 * *pData*, **IFX_UINT32** *bytesRequested*, **IFX_UINT8** *ZLP_flag*)

USB transmit function for EP0. This function is implemented as synchronous call. The application needs to call this for transmitting data on any end-point. The caller is blocked until return is executed. The function is not re-entrant and currently implemented as low-level driver call. Eventhough call is not re-entrant (global tx fifo filling warning flag used), even if a call comes during execution of this routine, since fifo validity check inside this funtion gets negated, this function will return error code.

Defined in: **usb_iil_api.h**

3.10.1 Return Value

usb status

USB_ERROR

if trasnmit request fails

Else

Actual number of bytes transmitted.

3.10.2 Parameters

pData

pointer to application data buffer

bytesRequested

Number of bytes requested by application for transmission

ZLP_flag

zero length packet flag. set this to 1 if zero packet length need to be sent.

3.10.3 Comments

This interface doesnot check whether the endpoint is configured for reception or transmission. So it is upto the application for raising the transmit request on proper endpoint number.

3.11 USBD_transmit

IFX_SINT32 USBD_transmit(IFX_UINT8 *ep*, **IFX_UINT8 *** *pData*, **IFX_UINT32** *bytesRequested*, **IFX_UINT8** *ZLP_flag*)

USB transmit function. This function is implemented as synchronous call. The application needs to call this for transmitting data on any end-point. The caller is blocked until return is executed. The function is not re-entrant and currently implemented as low-level driver call. Eventhough call is not re-entrant (global tx fifo filling warning flag used), even if a call comes during execution of this routine, since fifo validity check inside this funtion gets negated, this function will return error code.

Defined in: **usb_iil_api.h**

3.11.1 Return Value

usb status

USB_ERROR

if trasnmit request fails

Else

Actual number of bytes transmitted.

3.11.2 Parameters

ep

endpoint number for transmission

pData

pointer to application data buffer

bytesRequested

Number of bytes requested by application for transmission

ZLP_flag

zero length packet flag. set this to 1 if zero packet length need to be sent. Currently not used in low level.

3.11.3 Comments

This interface doesnot check whether the endpoint is configured for reception or transmission. So it is upto the application for raising the transmit request on proper endpoint number.

3.12 USBD_transmit_Auto_mode

IFX_SINT32 USBD_transmit_Auto_mode(IFX_UINT8 *ep*, **IFX_UINT8 *** *pData*, **IFX_UINT32** *bytesRequested*, **IFX_UINT8** *ZLP_flag*)

USB transmit function for automatic mode. This function is implemented as synchronous call. The application needs to call this for transmitting data on any end-point. The caller is blocked until return is executed. The function is not re-entrant and currently implemented as low-level driver call. Eventhough call is not re-entrant (global tx fifo filling warning flag used), even if a call comes during execution of this routine, since fifo validity check inside this funtion gets negated, this function will return error code.

Defined in: **usb_iil_api.h**

3.12.1 Return Value

usb status

USB_ERROR

if trasnmit request fails

Else

Actual number of bytes transmitted.

3.12.2 Parameters

ep
endpoint number for transmission

pData
pointer to application data buffer

bytesRequested
Number of bytes requested by application for transmission

ZLP_flag
zero length packet flag. set this to 1 if zero packet length need to be sent. Currently not used in low level.

3.12.3 Comments

This interface doesnot check whether the endpoint is configured for reception or transmission. So it is upto the application for raising the transmit request on proper endpoint number.

3.13 USB_receive

IFX_SINT32 USB_receive(IFX_UINT8 * *pData*, IFX_UINT16 *requestNbytes*, IFX_UINT8 *ep*)

USB receive function. This function is implemented as synchronous call. The application needs to call this for receving data on any end-point exculding EP0. The caller is blocked until some bytes get filled on the supplied end-point. If actual number of bytes received are lesser than the requested bytes then also this function gets unblocked and returns. If doesnot wait until requested number of bytes are received by low-level driver.

Defined in: **usb_iil_api.h**

3.13.1 Return Value

usb status

USB_ERROR
if receive request fails

Else
Actual number of bytes received

3.13.2 Parameters

pData

pointer to application data buffer.

requestNbytes

Number of bytes requested by application for reception.

ep

endpoint number for reception

3.13.3 Comments

This interface doenot check whether the endpoint is configured for reception or transmission. So it is upto the application for raising the receive request on proper endpoint number.

3.14 USBD_do_remote_wakeup

IFX_UINT32 USBD_do_remote_wakeup(void)

USB remote wakeup function. This function implements remote wakeup if the usb bus is in suspended mode. The application should check for bus suspension USB_SUSPENDED mode and then can do a call back to this interface.

Defined in: **usb_iil_api.h**

Return Value

Always 0

4 USB HAL Configuration

The configuration parameters defined should be correct for proper funtioning of USB HAL . The user needs to configure all the USB1.1 device frame work defintions in the file **usb_iil_cfg.h**. Since TC1130 also expects some of the device frame work configuration parameters related to end points, the user needs to define it in **usb_idl_cfg.h** In addition the user can configure the device endpoint properties, interrupts etc related to USB device in the file USB_IDL_CFG.H

The usb interrupts are fixed as on lines as below -

- USB device interrupts on node 0
- USB endpoint 1 to 5 on interrupt nodes 1 to 5 respectively
- USB endpoint 6 to 7 on interrupt node 6
- USB endpoint 8 to 11 on interrupt node 7

4.1 USB 1.1 specification parameters

In “USB enumeration definitions” section below all the structures accodring to USB1.1 are defined and in “USB enumeration initialization” section all these structures are filled.

4.1.1 USB ENUMERATION Definitions

All the set-up process descriptors according to USB1.1 specification are defined in this file usb_ill_setup.h. User need not modify any field here unless specified. The user can initialize all these descriptors in a separate file **usb_iil_cfg.h.**

4.1.2 USB_SETUP_REQUEST Structure

typedef struct {

} USB_setup_request;

setup request descriptor

Defined in: **usb_iil_setup.h**

4.1.3 USB_DEVICE_DESCRIPTOR_TYPE macro

```
#define USB_DEVICE_DESCRIPTOR_TYPE
```

Defined in: **usb_iil_setup.h**

Device descriptor constant

4.1.4 USB_CONFIGURATION_DESCRIPTOR_TYPE macro

```
#define USB_CONFIGURATION_DESCRIPTOR_TYPE
```

Defined in: **usb_iil_setup.h**

configuration descriptor constant

4.1.5 USB_STRING_DESCRIPTOR_TYPE macro

```
#define USB_STRING_DESCRIPTOR_TYPE
```

Defined in: **usb_iil_setup.h**

String descriptor constant

4.1.6 USB_INTERFACE_DESCRIPTOR_TYPE macro

```
#define USB_INTERFACE_DESCRIPTOR_TYPE
```

Defined in: **usb_iil_setup.h**

Interface descriptor constant

4.1.7 USB_ENDPOINT_DESCRIPTOR_TYPE macro

```
#define USB_ENDPOINT_DESCRIPTOR_TYPE
```

Defined in: **usb_iil_setup.h**

Endpoint descriptor constant

4.1.8 USB_DEVICE_DESCRIPTOR Structure

```
typedef struct {
IFX_UINT8 bLength;
IFX_UINT8 bDescriptorType;
IFX_UINT16 bcdUSB;
IFX_UINT8 bDeviceClass;
IFX_UINT8 bDeviceSubClass;
IFX_UINT8 bDeviceProtocol;
IFX_UINT8 bMaxPacketSize0;
IFX_UINT16 idVendor;
IFX_UINT16 idProduct;
IFX_UINT16 bcdDevice;
IFX_UINT8 iManufacturer;
IFX_UINT8 iProduct;
IFX_UINT8 iSerialNumber;
IFX_UINT8 bNumConfigurations;
} USB_DEVICE_DESCRIPTOR;
```

Defined in: **usb_iil_setup.h**

4.1.8.1 Members

bLength
size of this device descriptor in bytes

bDescriptorType
Device descriptor type: 01h const

bcdUSB
USB Spec release number: ver1.0 is 0100, ver1.1 is 0110

bDeviceClass
Class code: FFh means vendor specifc

bDeviceSubClass
Sub class code: FFh means vendor specifc

bDeviceProtocol
protocol code: FFh means vendor specifc

bMaxPacketSize0
max payload for EP0

idVendor
Vendor ID assigned by USB forum

idProduct
Product ID assigned by Manufacturer

bcdDevice
Device Release number in BCD

iManufacturer
Index of String descriptor describing Manufacturer

iProduct
Index of String descriptor describing Product

iSerialNumber
Index of String descriptor describing device's Serial number

bNumConfigurations
Number of possible configurations on this device

4.1.8.2 Comments

The usb device descriptor. For initialization see CONFIG_USB_DEVICE_DESCRIPTOR macro. If any field is changed like iProduct ..., the application should take care to modify the **usb_iil_setup.c** file accordingly.

4.1.9 USB_ENDPOINT_DESCRIPTOR Structure

```
typedef struct {
                                        IFX_UINT8 bLength;
                                        IFX_UINT8 bDescriptorType;
                                        IFX_UINT8 bEndpointAddress;
                                        IFX_UINT8 bmAttributes;
                                        IFX_UINT8 wMaxPacketSizeLSB;
                                        IFX_UINT8 wMaxPacketSizeMSB;
                                        IFX_UINT8 bInterval;
} USB_ENDPOINT_DESCRIPTOR;
```

USB endpoint descriptor All the endpoint descriptors declared in **usb_iil_setup.h** should use the following fields.

Defined in: **usb_iil_setup.h**

4.1.9.1 Members

bLength
size of thid descriptor in bytes

bDescriptorType
endpoint descriptor type- 05h const

bEndpointAddress
endpoint number & direction; "Bit 7: Direction(0:OUT, 1: IN),Bit 6-4:Res,Bit 3-0:EP no"

bmAttributes
Bit 1-0: Transfer Type 00:Control,01:Isochronous,10: Bulk, 11: Interrupt

wMaxPacketSizeLSB
LSB of Maximum Packet Size for this EP

wMaxPacketSizeMSB
MSB of Maximum Packet Size for this EP

bInterval
Interval for polling EP, applicable for Interrupt and Isochronous data transfer.

4.1.9.2 Comments

Not to be confused with on-chip USB device endpoints.

4.1.10 USB_CONFIGURATION_DESCRIPTOR Structure

```
typedef struct {
                                        IFX_UINT8 bLength;
                                        IFX_UINT8 bDescriptorType;
                                        IFX_UINT8 wTotalLengthLSB;
                                        IFX_UINT8 wTotalLengthMSB;
                                        IFX_UINT8 bNumInterfaces;
                                        IFX_UINT8 bConfigurationValue;
                                        IFX_UINT8 iConfiguration;
                                        IFX_UINT8 bmAttributes;
                                        IFX_UINT8 MaxPower;
} USB_CONFIGURATION_DESCRIPTOR;
```

USB device configuration descriptor. The application can specify the device capablities like remote wake-up, power consumption, number of configuration and interfaces supported.

Defined in: **usb_iil_setup.h**

4.1.10.1 Members

bLength
size of thid descriptor in bytes

bDescriptorType
Configuration descriptor type - 02h const

wTotalLengthLSB
LSB Total length of data for this configuration

wTotalLengthMSB
MSB Total length of data for this configuration

bNumInterfaces
Nos of Interface supported by this configuration

bConfigurationValue
Identifies the configuration in Get_configuration & Set_Configuration

iConfiguration
Index to the string that describes the configuration

bmAttributes
Config Characteristcs bitmap "D7,D4-D0: Res, D6: Self Powered,D5: Remote Wakeup

MaxPower
Max Power Consumption of the USB device

4.1.10.2 Comments

For initialization see CONFIG_USB_CONFIGURATION_DESCRIPTOR

4.1.11 USB_INTERFACE_DESCRIPTOR Structure

```
typedef struct {
                                        IFX_UINT8 bLength;
                                        IFX_UINT8 bDescriptorType;
                                        IFX_UINT8 bInterfaceNumber;
                                        IFX_UINT8 bAlternateSetting;
                                        IFX_UINT8 bNumEndpoints;
                                        IFX_UINT8 bInterfaceClass;
                                        IFX_UINT8 bInterfaceSubClass;
                                        IFX_UINT8 bInterfaceProtocol;
                                        IFX_UINT8 iInterface;
} USB_INTERFACE_DESCRIPTOR;
```

USB interface descriptor All the interfaces descriptors declared in **usb_iil_setup.h** should use the following fields.

Defined in: **usb_iil_setup.h**

4.1.11.1 Members

bLength
size of thid descriptor in bytes

bDescriptorType
Interface Descriptor Type - 04h const

bInterfaceNumber
Number of Interface

bAlternateSetting
Value used to select alternate setting

bNumEndpoints
Nos of Endpoints used by this Interface

bInterfaceClass
Class code assigned bu USB Forum "0xFF Vendor specific"

bInterfaceSubClass
Subclass code assigned bu USB Forum "0xFF Vendor specific"

bInterfaceProtocol
Protocol code assigned bu USB Forum "0xFF Vendor specific"

iInterface
Index of String descriptor describing Interface

4.1.12 MANUFACTURER_STRING_LENGTH macro

```
#define MANUFACTURER_STRING_LENGTH
```

Defined in: **usb_iil_setup.h**

Maximum string length for manufacturer descritpor.

4.1.12.1 Comments

Change this if any modification is done to Manufacturer_Descriptor structure.

4.1.13 USB_MANUFACTURER_DESCRIPTOR Structure

```
typedef struct {
                                        IFX_UINT8 bLength;
                                        IFX_UINT8 bDescriptorType;
                                        IFX_UINT8
bString[MANUFACTURER_STRING_LENGTH];
} USB_MANUFACTURER_DESCRIPTOR;
```

The application can change the manufacturer string here. Currently set to "Infineon Technologies". See Manufacturer_Descriptor to initialize according to requirement.

Defined in: **usb_iil_setup.h**

4.1.13.1 Members

bLength
size of this descritor

bDescriptorType
STRING descriptor type.

bString[MANUFACTURER_STRING_LENGTH]
string to be passed or captured

4.1.14 USBD_LANG_DESCRIPTOR Structure

```
typedef struct {
                                        IFX_UINT8 bLength;
                                        IFX_UINT8 bDescriptorType;
                                        IFX_UINT16 wLANGID0;
} USBD_LANG_DESCRIPTOR;
```

Language descriptor. The application can change the language type here. Currently set to US ENGLISH. See LANGID_Descriptor to initialize according to requirement.

Defined in: **usb_iil_setup.h**

4.1.14.1 Members

bLength
size of this descritor

bDescriptorType
STRING descriptor type.

wLANGID0
Language Id

4.1.15 PRODUCT_STRING_LENGTH macro

```
#define PRODUCT_STRING_LENGTH
```

Defined in: **usb_iil_setup.h**

Maximum string length used in USB_PRODUCT_STRING descriptor.

4.1.15.1 Comments

change this if string length exceeds 25 in Product_Descriptor.

4.1.16 USB_PRODUCT_STRING Structure

```
typedef struct {
                                        IFX_UINT8 bLength;
                                        IFX_UINT8 bDescriptorType;
                                        IFX_UINT8
bString[PRODUCT_STRING_LENGTH];
} USB_PRODUCT_STRING;
```

USB product string. Currently initialized to "TC1130 USB" .

Defined in: **usb_iil_setup.h**

4.1.16.1 Members

bLength
size of this descritor

bDescriptorType
STRING descriptor type.

bString[PRODUCT_STRING_LENGTH]
string to be passed or captured

4.1.16.2 Comments

See Product_Descriptor to initialize according to requirement.

4.1.17 SERIAL_STRING_LENGTH macro

```
#define SERIAL_STRING_LENGTH
```

Defined in: **usb_iil_setup.h**

Maximum serial number length for USB_SERIAL_NUMBER_DESCRIPTOR.

4.1.17.1 Comments

Change this if any modification is done to USB_Serial_number_descriptor structure.

4.1.18 USB_SERIAL_NUMBER_DESCRIPTOR Structure

```
typedef struct {
                                        IFX_UINT8 bLength;
                                        IFX_UINT8 bDescriptorType;
                                        IFX_UINT8
bString[SERIAL_STRING_LENGTH];
} USB_SERIAL_NUMBER_DESCRIPTOR;
```

Device/Product serial number. Currently initialized to "TC1130A USB".

Defined in: **usb_iil_setup.h**

4.1.18.1 Members

bLength
size of this descritor

bDescriptorType
STRING descriptor type.

bString[SERIAL_STRING_LENGTH]
string to be passed or captured

4.1.18.2 Comments

See USB_Serial_number_descriptor to initialize accordingly.

4.1.19 USB ENUMERATION initializations

All the set-up process descriptors according to USB1.1 specifications described above are initialized in the file **usb_iil_cfg.h**. The application can configure this file according to requirement. Also see file **usb_iil_setup.h** file for all the field definitions. For more details please refer to USB1.1 specifications.

4.2 USB ENUMERATION CONSTANTS in file usb_iil_cfg.h

All the set-up process descriptors according to USB1.1 specifications are initialized in this file **usb_iil_cfg.h**. The application can configure this file according to requirement. Also see file **usb_iil_setup.h** file for all the field definitions. For more details please refer to USB1.1 specifications.

4.3 Device descriptor initialization

4.3.1 CONFIG_USB_DEVICE_DESCRIPTOR macro

```
#define CONFIG_USB_DEVICE_DESCRIPTOR
```

Defined in: **usb_iil_cfg.h**

Device descriptor according to USB_DEVICE_DESCRIPTOR fields.

4.4 Configuration descriptor initialization

4.4.1 CONFIG_USB_NUMBER_OF_CONFIGURATIONS macro

```
#define CONFIG_USB_NUMBER_OF_CONFIGURATIONS
```

Defined in: **usb_iil_cfg.h**

Number of configurations supported by this USB device. EP0 is supported on configuration 0 and all others are on CF1.

4.4.2 CONFIG_USB_CONFIGURATION_DESCRIPTOR macro

```
#define CONFIG_USB_CONFIGURATION_DESCRIPTOR
```

Defined in: **usb_iil_cfg.h**

Configuration descriptor according to USB_CONFIGURATION_DESCRIPTOR fields. The user needs to change the wTotalLenght field if CONFIG_USB_TOTBYTES macro below changes.Also if any change is done to bNumInterfaces, change the CONFIG_USB_TOTAL macro accordingly. Currenlty there are four interfaces supported by this device.

4.5 Interface 0 initialization

4.5.1 CONFIG_USB_INTERFACE_DESCRIPTOR_IF0AS0 macro

```
#define CONFIG_USB_INTERFACE_DESCRIPTOR_IF0AS0
```

Defined in: **usb_iil_cfg.h**

Interface-0 AlternateSetting-0 descriptor according to USB_INTERFACE_DESCRIPTOR fields. The user need not change any field here.

4.5.2 CONFIG_USB_INTERFACE_DESCRIPTOR_IF0AS1 macro

#define CONFIG_USB_INTERFACE_DESCRIPTOR_IF0AS1

Defined in: **usb_iil_cfg.h**

4.5.3 CONFIG_USB_INTERFACE_DESCRIPTOR_IF1AS0 macro

```
#define CONFIG_USB_INTERFACE_DESCRIPTOR_IF1AS0
```

Defined in: **usb_iil_cfg.h**

4.5.4 CONFIG_USB_INTERFACE_DESCRIPTOR_IF1AS1 macro

```
#define CONFIG_USB_INTERFACE_DESCRIPTOR_IF1AS1
```

Defined in: **usb_iil_cfg.h**

4.5.5 CONFIG_USB_INTERFACE_DESCRIPTOR_IF1AS1EP1 macro

```
#define CONFIG_USB_INTERFACE_DESCRIPTOR_IF1AS1EP1
```

Defined in: **usb_iil_cfg.h**

4.5.6 CONFIG_USB_INTERFACE_DESCRIPTOR_IF1AS1EP2 macro

```
#define CONFIG_USB_INTERFACE_DESCRIPTOR_IF1AS1EP2
```

Defined in: **usb_iil_cfg.h**

4.5.7 CONFIG_USB_INTERFACE_DESCRIPTOR_IF2AS0 macro

```
#define CONFIG_USB_INTERFACE_DESCRIPTOR_IF2AS0
```

Defined in: **usb_iil_cfg.h**

4.5.8 CONFIG_USB_INTERFACE_DESCRIPTOR_IF2AS1 macro

```
#define CONFIG_USB_INTERFACE_DESCRIPTOR_IF2AS1
```

Defined in: **usb_iil_cfg.h**

4.5.9 CONFIG_USB_INTERFACE_DESCRIPTOR_IF3AS0 macro

```
#define CONFIG_USB_INTERFACE_DESCRIPTOR_IF3AS0
```

Defined in: **usb_iil_cfg.h**

4.5.10 CONFIG_USB_INTERFACE_DESCRIPTOR_IF3AS1 macro

```
#define CONFIG_USB_INTERFACE_DESCRIPTOR_IF3AS1
```

Defined in: **usb_iil_cfg.h**

4.6 All descriptors

4.6.1 CONFIG_USB_TOTAL macro

```
#define CONFIG_USB_TOTAL
```

Defined in: **usb_iil_cfg.h**

The CONFIG_USB_TOTAL contains all the descriptors supported by this device. i.e., Configuration + Interface0+ Interface1+ interface2 + Interface3 defined above. Place the descriptors in order. i.e,Interface number - its alternate setting - endpoints belonging to this alternate setting. Currently the device driver doesnot support alternate setting AS2/AS1 on same interface. If you alter this configuration make sure to alter CONFIG_USB_TOTBYTES. Also the USBD (usb device) configuration needs to be changed in **usb_idl_cfg.h** file which is spcecific to TC1130.

4.6.2 CONFIG_USB_TOTBYTES macro

```
#define CONFIG_USB_TOTBYTES
```

Defined in: **usb_iil_cfg.h**

CONFIG_USB_TOTBYTES contains total number of bytes in CONFIG_USB_TOTAL. Calculation is as follows. Total Bytes (Number of endpoint descriptor defined in CONFIG_USB_TOTAL * 7) + (Number of other descriptors * 9).

4.7 String descriptors Initialization

4.7.1 LANGID_Descriptor Structure

```
typedef struct {
} LANGID_Descriptor;
```

Holds the initialization parameters for language id descriptor USBD_LANG_DESCRIPTOR. Currnetly set for US english.

Defined in: **usb_iil_cfg.h**

4.7.2 Manufacturer_Descriptor Structure

```
typedef struct {
                                        sizeof;
                                        USB_STRING_DESCRIPTOR_TYPE;
} Manufacturer_Descriptor;
```

Holds the initialization parameters for manufacturer descriptor.

Defined in: **usb_iil_cfg.h**

4.7.2.1 Members

sizeof
bLength

USB_STRING_DESCRIPTOR_TYPE
bDescriptorType

4.7.2.2 Comments

The user can change the manufacturer string here, but should remember to change the MANUFACTURER_STRING_LENGTH field. Currently MANUFACTURER_STRING_LENGTH is set to 50.

4.7.3 Product_Descriptor Structure

```
typedef struct {
                                        sizeof;
                                        USB_STRING_DESCRIPTOR_TYPE;
} Product_Descriptor;
```

Holds the initialization parameters for manufacturer descriptor.

Defined in: **usb_iil_cfg.h**

4.7.3.1 Members

sizeof
bLength

USB_STRING_DESCRIPTOR_TYPE
bDescriptorType

4.7.3.2 Comments

The user can change the product string (TC1130 USB) here, but remember to change the PRODUCT_STRING_LENGTH field. Currently PRODUCT_STRING_LENGTH is set to 30.

4.7.4 USB_Serial_Descriptor Structure

```
typedef struct {
                                        sizeof;
                                        USB_STRING_DESCRIPTOR_TYPE;
} USB_Serial_Descriptor;
```

Holds the initialization parameters for product serial number.

Defined in: **usb_iil_cfg.h**

4.7.4.1 Members

sizeof
bLength

USB_STRING_DESCRIPTOR_TYPE
bDescriptorType

4.7.4.2 Comments

The user can change the serial string here, but remember to change the SERIAL_STRING_LENGTH field also. Currently SERIAL_STRING_LENGTH is set to 30.

4.7.5 USBD_CFG_DEVICE_FULL_SPEED macro

```
#define USBD_CFG_DEVICE_FULL_SPEED
```

Defined in: **usb_idl_cfg.h**

This will hold the full speed or low speed device definition. Equate this to 1 for FULL speed and 0 for LOW speed device.

4.7.6 USBD_CFG_TOTAL_EP_USED macro

```
#define USBD_CFG_TOTAL_EP_USED
```

Defined in: **usb_idl_cfg.h**

Declare the maximum endpoints (logical eps) used by this device including EP0. This defintion will be common to GLOBAL_USBD_CFG_BLOCKS, GLOBAL_USBD_CFG_BUFFER & endpoint interrupts definitions.

4.7.7 USBD_MANUAL_MODE/USBD_AUTO_MODE macro

```
#define USBD_MANUAL_MODE
```

Defined in: **usb_idl_cfg.h**

This definition makes the USB LLD to work in manual mode, and the USBD_AUTO_MODE, switches the mode to automatic mode

Note: Automatic mode can ba tested only with the TC1130 BB Step board

4.7.8 GLOBAL_USBD_CFG_BLOCKS macro

```
#define GLOBAL_USBD_CFG_BLOCKS
```

Defined in: **usb_idl_cfg.h**

This will hold 6-Byte USB Configuration block for all EPs. Add all the defined USBD_CFG_BLOCK_n blocks here.

4.7.9 GLOBAL_USBD_CFG_BLOCKS macro

```
#define GLOBAL_USBD_CFG_BLOCKS
```

Defined in: **usb_idl_cfg.h**

Add all the defined USBD_CFG_BLOCK_n blocks here.

4.7.10 USBD_CFG_BLOCK_LENGTH macro

```
#define USBD_CFG_BLOCK_LENGTH
```

Defined in: **usb_idl_cfg.h**

Change this value according to the maximum number of USBD_CFG_BLOCK_n defined in GLOBAL_USBD_CFG_BLOCKS. User need not modify this value. 6 bytes for each block.

4.7.11 USBD_CFG_BUFFER_EPx[0-11] macro

```
#define USBD_CFG_BUFFER_EPx[0-11]
```

Defined in: **usb_idl_cfg.h**

Configure the buffer area for endpoint n The fisrt parameter is buffer size & second is number of buffers. The total size is each buffer size multiplied by number of buffers. Donot change the configuration of EP0.

4.7.12 USBD_CFG_BUFFER_EP EP_0n macro

```
#define USBD_CFG_BUFFER_EP EP_0n
```

Defined in: **usb_idl_cfg.h**

4.7.13 GLOBAL_USBD_CFG_BUFFER macro

```
#define GLOBAL_USBD_CFG_BUFFER
```

Defined in: **usb_idl_cfg.h**

Holds all the cofigured buffer properties for all endpoints. The total hardware fifo size avaiable is 0x65f (1631 bytes). So the size of this buffer should not exceed this limit. Also note that between each endpoint the low-level driver allocates an extra 8 bytes.

4.7.14 USBD_CFG_BLOCK_LENGTH macro

```
#define USBD_CFG_BLOCK_LENGTH
```

Defined in: **usb_idl_cfg.h**

Holds the lenght of total bytes used in GLOBAL_USBD_CFG_BUFFER. User need not modify this value. 2 bytes for each buffer.

Configure the end point properties definitions. All device interrupts are on source node 0. EP1 on source node 1, EP2 on source node 2, EP3 on source node 3, EP4 on source node 4, EP5 on source node 5, EP6-EP7 on source node 6, EP8-EP9-EP10-EP11 are on source node 7. Non supported EPs should be filled with value 0x00.

4.7.15 USBD_CFG_DEVICE_INTERRUPT macro

```
#define USBD_CFG_DEVICE_INTERRUPT
```

Defined in: **usb_idl_cfg.h**

This will hold all device interrupts for DIER register. This will also hold interrupt source node of device interrupts. Currently all device interrupts are configured on source node 0.

4.7.16 USBD_CFG_EP_INTERRUPT_GLOBAL macro

```
#define USBD_CFG_EP_INTERRUPT_GLOBAL
```

Defined in: **usb_idl_cfg.h** : Holds all EP interrupt values for EPICn register. Drivers will use upto maximum EPs configures - USBD_CFG_TOTAL-EP_USED(Non modifiable)

5 Application Examples

This section presents a simple example application for using the USB HAL.

The below code shows how to initialize the usb device. The code shows reception of 64 bytes on end-point 2 and transmission of received bytes on end-point number 1.

5.1 Example for non-OS implementations

```
#include "compiler.h"
#include "usb_iil_api.h"
#include "sys_api.h"
#include "usb_idl_cfg.h"


#ifdef IFX_COMPILER_GNU
static IFX_UINT8 main_buffer[1200] __attribute__ ((aligned (4)));
#else   /*for tasking*/
#pragma align 4
static IFX_UINT8 main_buffer[1200];
#ifdef IFX_TASKING_VER1
#pragma align off
#endif
#ifdef IFX_TASKING_VER2
#pragma align restore
#endif

#endif /* of IFX_COMPILER_GNU */


#define P0_DIR                  (*((IFX_UINT32 volatile *) 0xF0000C18))
#define P0_OUT                  (*((IFX_UINT32 volatile *) 0xF0000C10))

int main( void )
{
IFX_UINT8 *pData = main_buffer;
IFX_SINT16 nBytesRequest, nBytesReceived;
IFX_UINT32 status;
#define RX_PIPE 2
#define TX_PIPE 1


              DISABLE_GLOBAL_INTERRUPT();
              /* Initialize USB clock. */
```

```
                SYS_clk_initialise();
                for(status = 1; status<1200;status++, pData++)
                {
                 *pData = (IFX_UINT8)status;
                }
                pData = main_buffer;

                while( USBD_device_initialize() != USB_OK );      /* Init
error */
                ENABLE_GLOBAL_INTERRUPT();


    //LED init
    P0_DIR = 0x080;
    P0_OUT = 0x080;

    while (1)
    {
                status = USBD_get_device_state();
                switch(status)
                {
                case(USB_CONFIGURED):
                P0_OUT = 0x000;
                /*Max request can be 1023 bytes only*/
                nBytesRequest = 64;

              nBytesReceived  = USB_receive(pData, nBytesRequest, RX_PIPE
);
                if(nBytesReceived > 0)
                                {
                        nBytesReceived = USBD_transmit( TX_PIPE,  pData,
nBytesReceived, 0);
                      }
                break;
                default:
                break;
               }
    }

return 0;
} /* end of main */
```

```
void Sys_BackGround(void)
{
//switch the LED off in case the USB device is Detached
if (USBD_get_device_state() != USB_CONFIGURED)P0_OUT = 0x080;
}
```

5.2 Example for DAS Application

```
#include "compiler.h"
#include "usb_iil_api.h"
#include "usb_idl_cfg.h"
#include "sys_api.h"
#include "usbd_idl_macro.h"

#ifdef IFX_COMPILER_GNU
static IFX_UINT8 main_buffer[520] __attribute__ ((aligned (4)));
static IFX_UINT8 main_buffer2[520] __attribute__ ((aligned (4)));
static IFX_UINT8 main_buffer3[64] __attribute__ ((aligned (4)));

#else   /*for tasking*/
#pragma align 4
static IFX_UINT8 main_buffer[520];
static IFX_UINT8 main_buffer2[520];
static IFX_UINT8 main_buffer3[64];

#ifdef IFX_TASKING_VER1
#pragma align off
#endif
#ifdef IFX_TASKING_VER2
#pragma align restore
#endif

#endif /* of IFX_COMPILER_GNU */


extern void *copy(void *dest, const void *src, IFX_UINT32 len);

#define P0_DIR              (*((IFX_UINT32 volatile *) 0xF0000C18))
#define P0_OUT               (*((IFX_UINT32 volatile *) 0xF0000C10))

int main( void )
{
IFX_UINT8 *pData = main_buffer;
IFX_UINT8 *pData2 = main_buffer2;
```

```
IFX_UINT8 *pData3 = main_buffer3;
//IFX_UINT32 c;

IFX_SINT16 nBytesRequest, nBytesReceived=0;
#ifdef USBD_AUTO_MODE
IFX_SINT16 nBytesTransmitted;
//IFX_SINT32 loopend=100000;

#endif

#ifdef USBD_MANUAL_MODE
IFX_UINT32 i;
#endif

IFX_UINT32 status;
static IFX_UINT16 total_length, expected_das_pkt_length, expected_das_pkt_length2 ;
//static IFX_UINT16 second_das_pkt_length=0;
//static IFX_UINT16 pi;
static IFX_UINT16 old_expected_das_pkt_length, size_already_found, free_mainbuffer2;
IFX_UINT16 FillSize;
#define RX_PIPE 2
#define TX_PIPE 1


  DISABLE_GLOBAL_INTERRUPT();

  // Initialize USB clock
  SYS_clk_initialise();

  // Initialise main_buffers
  for(status = 1; status<520;status++, pData++)
  {
    *pData = (IFX_UINT8)status;
  }
  pData = main_buffer;

  for(status = 1; status<520;status++, pData2++)
  {
    *pData2 = (IFX_UINT8)status;
  }
  pData2 = main_buffer2;

  for(status = 1; status<64;status++, pData3++)
  {
    *pData3 = (IFX_UINT8)status;
  }
  pData3 = main_buffer3;
```

```
    while( USBD_device_initialize() != USB_OK );     // Init error

    ENABLE_GLOBAL_INTERRUPT();

    total_length = 0;
    expected_das_pkt_length = 0;
    old_expected_das_pkt_length = 0;
    size_already_found=0;
    free_mainbuffer2 = 0;
    expected_das_pkt_length2=0;
    FillSize =0;
    // LED init
    P0_DIR = 0x080;
    // Switch off yellow LED
    P0_OUT = 0x080;


    while (1)
    {
          // Get USB_11D device status
     status = USBD_get_device_state();

     switch(status)
          {
            case(USB_CONFIGURED):

       // Switch on yellow LED
          P0_OUT = 0x000;

//#####################################################################
// Collect DAS Message
//#####################################################################


////###################################################################
// Routine for AUTOMATIC MODE
// for TC1130 BB steps only
////###################################################################

#ifdef USBD_AUTO_MODE

//Max request can be 1023 bytes only
nBytesRequest   = 64;
nBytesReceived  = 0;

// Get 64 bytes from FIFO as long as device is configured
```

```
if (!free_mainbuffer2){

   // Check of NEB(ep < BSZ(ep)!  Mandatory!
  IFX_UINT32 *p32_epp;
                    p32_epp = (IFX_UINT32 *)( (IFX_UINT8 *)USBD_EPUP_BASE + (2 * 0x0010) );
                    if((p32_epp[3]>>16) < 6) {
   // Get 64 bytes from FIFO as long as device is configured
      nBytesReceived  = USB_receive(pData, nBytesRequest, RX_PIPE );
  }
}

// Check received message for DAS header, especially for size
if (nBytesReceived){
  if (!size_already_found){

    // Net length = received bytes
    total_length = nBytesReceived;

    // Reset Main Buffer since header packet arrived
    pData = main_buffer;

    // Get the length of DAS Message
    copy(&expected_das_pkt_length, main_buffer, 2);

    // Debug stuff
    if (expected_das_pkt_length > 520){
      P0_OUT = 0x080;
    }

    // ShortPacket Check I (Packet < 64 byte)
    if ((expected_das_pkt_length%64)&&(expected_das_pkt_length<64)){
      //DAS Header + Short Package
      total_length=expected_das_pkt_length;
      pData+= total_length;
    } else {
      pData+= nBytesReceived;
    }

    // Enable "Collect following packets" routine
    size_already_found=1;

     } else {
         total_length += nBytesReceived;
  // ShortPacket Check II (Packet > 64 byte)
  if (total_length > expected_das_pkt_length){
    total_length = expected_das_pkt_length;
  }
```

```
        // Increase pData pointer to next free byte
  pData += nBytesReceived;
  }
      }

//#####################################################################
// COPY collected DAS Message from main_buffer to main_buffer2
//#####################################################################

            if ((total_length!=0)&&(total_length >= expected_das_pkt_length)&&(!free_mainbuffer2)) {

// Reset main_buffer
pData = main_buffer;

// Copy the length of DAS Message to main_buffer2 variable
expected_das_pkt_length2=expected_das_pkt_length;

//reset length value for main_buffer
expected_das_pkt_length=0;

// Copy mainBuffer to mainBuffer2
      for(status = 0; status<expected_das_pkt_length2;status++, pData2++){
                        *pData2 = pData[status];
                       }

// Reset main_buffers
      pData = main_buffer;
      pData2 = main_buffer2;

// Change third byte from 0x4e to 0xce
pData2[2] = 0xCE;

// Reset main_buffers
      pData2 = main_buffer2;

      // DONE Bit OR Zero Length Packet after Transmit ?
      FillSize = (expected_das_pkt_length2 % 64);

      // Enable main_buffer2: Ready to transmit!
      free_mainbuffer2=1;

      // Enable DAS Msg Collect routine for next DAS Header
      size_already_found=0;

// Reset total_length
total_length = 0;
```

```
}

//####################################################################
// Transmit DAS Message from main_buffer2
//####################################################################

if (free_mainbuffer2){

 IFX_UINT32 Transmitted_Bytes=0 ;


// Trigger Single USB Transmit of 64 or less bytes
if (expected_das_pkt_length2 >= 64){
  nBytesTransmitted = USBD_transmit_Auto_mode( TX_PIPE, pData2, 64, 0);
  Transmitted_Bytes+=nBytesTransmitted;
} else {
  nBytesTransmitted = USBD_transmit_Auto_mode( TX_PIPE, pData2, expected_das_pkt_length2, 0);
  Transmitted_Bytes+=64;
}

// Recalculate leaving DAS message-length
      expected_das_pkt_length2-=nBytesTransmitted;
pData2+=nBytesTransmitted;

// Send Zero Length Packet, when Size%64 = 0
// Transfer ends with even packetsize = 64
// This signals an end of transfer to the Host

      if ((expected_das_pkt_length2==0)&&(!FillSize)) {
        free_mainbuffer2=0;
  pData2 = main_buffer2;
        USBD_TX_ZERO_LENGTH_PKT(TX_PIPE)
      }

      // Write DONE-Bit, when Size%64 != 0
// Transfer ends with odd paketsize =< 64
      // This signals an end of transfer to the Host

if ((expected_das_pkt_length2==0)&&(FillSize)) {
        free_mainbuffer2=0;
  pData2 = main_buffer2;
        USBD_SET_DONE_BIT(TX_PIPE);
      }
}

#endif
```

```
////#####################################################################
// Routine for MANUAL MODE
// for TC1130 AA and BA steps
////#####################################################################



#ifdef USBD_MANUAL_MODE
//Max request can be 1023 bytes only
nBytesRequest = 64;
// Get 64 bytes from FIFO as long device is configured
while (((nBytesReceived  = USB_receive(pData, nBytesRequest, RX_PIPE )) < 2) && (USBD_get_device_state()
== USB_CONFIGURED));

        // Get USB_11D device status, if USB_receive loop is exit with a non confugured state, stop the case
        if (USBD_get_device_state() != USB_CONFIGURED)break;

// DAS ECF changes
// Update current length of DAS packet
total_length = nBytesReceived;

// Get real length of DAS packet (first two bytes of a DAS packet)
pData = main_buffer;

copy(&expected_das_pkt_length, main_buffer, 2);

// Collect complete DAS packet
while (total_length < expected_das_pkt_length)
{
// Update pdata pointer with last read length
pData += nBytesReceived;
// Get 64 bytes from FIFO
nBytesReceived  = USB_receive(pData, nBytesRequest, RX_PIPE );
// Update of current length variable
total_length += nBytesReceived;
}

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;
          }
        }

           pData = main_buffer;

while (expected_das_pkt_length>0)
{
   nBytesReceived = USBD_transmit( TX_PIPE, pData, 64,0);
   expected_das_pkt_length-=nBytesReceived;
   pData+=nBytesReceived;
}
// Reset main_buffer
pData = main_buffer;

                    #endif

break;


      default:
         break;

        }
   }

return 0;

} /* end of main */

/*============================================================================
 * Function Name :: copy()
 *
 * Purpose       :: Copy the memory in the fifo
 *
 * Input         :: destination pointer, Source pointer, Length
 *
*============================================================================*/
void *copy(void *dest, const void *src, IFX_UINT32 len)
{
  IFX_UINT8 *dstp = (IFX_UINT8*)dest;
  IFX_UINT8 *srcp = (IFX_UINT8*)src;
  IFX_UINT32 i;

  for (i = 0; i < len; ++i)
  {
```

```
    dstp[i] = srcp[i];
  }
  return dest;
}
```

6 Application note on the disabling of interrupts in the Interrupt Service Routine

From the hardware perspective, an Interrupt Service Routine(ISR) is entered with the interrupt system globally disabled. The Low Level Driver(LLD) does not enable global interupt in the ISR, as the LLD ISRs are kept short. Most LLD ISRs invoke a callback function that was registered by the application. If required, the application may enable global interrupts (by calling ENABLE_GLOBAL_INTERRUPT()) at the beginning of the ISR callback function.

7 Related reference documents

- Infineon Technologies HAL/Device Driver Software Suite Overview
- USB LLD design document version 2.0 (USB_LLD_Design.pdf)
- TC1130 System Manual
- TC1130 Peripheral Manual
- USB 1.1 Specification Document
- USB Firmware Implementation Guideline

8 Appendix A - Infineon IFX types

To overcome the problem of the size of data types changing between different compilers the HAL software modules use IFX types. These are defined in a file called COMPILER.H which is generated for each compiler that is supported. Table 1 presents these IFX types.

Table 1 Table of IFX Data Types

IFX_UINT8	Unsigned 8 bit integer
IFX_UINT16	Unsigned 16 bit integer
IFX_UINT32	Unsigned 32 bit integer
IFX_SINT8	Signed 8 bit integer
IFX_SINT16	Signed 16 bit integer
IFX_SINT32	Signed 32 bit integer
IFX_VUINT8	Unsigned 8 bit volatile integer
IFX_VUINT16	Unsigned 16 bit volatile integer
IFX_VUINT32	Unsigned 32 bit volatile integer
IFX_VSINT8	Signed 8 bit volatile integer
IFX_VSINT16	Signed 16 bit volatile integer
IFX_VSINT32	Signed 32 bit volatile integer
IFX_SFLOAT	Signed flaot
IFX_STINT8	Signed static 8 bit integer
IFX_STINT16	Signed static 16 bit integer
IFX_STINT32	Signed static 32 bit integer
IFX_STUINT8	Unsigned static 8 bit integer
IFX_STUINT16	Unsigned static 16 bit integer
IFX_STUINT32	Unsigned static 32 bit integer

9 Appendix B - The SYS HAL Configurable Parameters

This section defines the configurable parameters of the SYS HAL - interrupts, GPIO ports, and the clock. The user may change only the value associated with the macros to suit application requirements. However, the user may NOT change the name of the macro.

9.1 System Clock Frequency

The clock must be operational before the controller can function. This clock is connected to the peripheral clock control registers, so changing the value of this clock frequency will affect all peripherals. The individual peripherals can scale down this frequency according to their requirements, for more details please refer to the corresponding user guide documents.

9.1.1 SYS_CLOCK_XXMHz macro

```
#define SYS_CLOCK_XXMHz
```

Defined in: **sys_cfg.h**

This is used for the user to configure for the USB Sytem clock frequency to either 48, or 96 or 144 MHz. Accordingly default values of the N,P,K values are configured to set the system clock frequency.

9.1.2 SYS_CFG_USB_DEVICE_ENABLE macro

```
#define SYS_CFG_USB_DEVICE_ENABLE
```

Defined in: **sys_cfg.h**

User needs to configure whether the USB device has been used.

1

Equate this macro to 1 if onchip USB device is used.

0

Equate this macro to 0 if usb device is not used (default).

9.1.3 SYS_CFG_USB_ONCHIP_CLK macro

```
#define SYS_CFG_USB_ONCHIP_CLK
```

Defined in: **sys_cfg.h**

User can configure the USB clock generation logic whether it internal or external. If clock is external, it will be derived from pin P4.0.

1

Equate this macro to 1 for internal clock generation

0

Equate this macro to 0 for external clock generation

9.1.3.1 Comments

Note that if the device is used and usb clock generation is internal, then cpu frequency should be programmed either to 48MHZ, 96MHZ or 144MHz.

9.1.4 SYS_CFG_USB_CLK_DIVISOR macro

```
#define SYS_CFG_USB_CLK_DIVISOR
```

Defined in: **sys_cfg.h**

User needs to configure the USB clock ratio based upon the USB clock frequency. Since clock frequency can be either 48 MHZ, 96 or 144 MHZ, the ratio can 1, 2 or 3 respectively.

9.1.5 SYS_CFG_OSC_FREQ macro

```
#define SYS_CFG_OSC_FREQ
```

Defined in: **sys_cfg.h**

User has to configure this with external applied frequency.

9.1.6 SYS_CFG_CLK_MODE macro

```
#define SYS_CFG_CLK_MODE
```

Defined in: **sys_cfg.h**

User needs to configure this macro to any one of the following clock operation mode.

0

Direct drive (CPU clock directly derived from external applied frequency, N, P, and K values are not considered).

1

PLL mode (N, P, K values will be considered to derive CPU clock frequency from external frequency)

2

VCO bypass/pre-scalar mode (N value not considered to derive CPU clock from external frequency).

9.1.7 SYS_CFG_FREQ_SEL macro

```
#define SYS_CFG_FREQ_SEL
```

Defined in: **sys_cfg.h**

This define decide the frequency ration between CPU and system, this is independent from the clock mode selection(SYS_CFG_CLK_MODE).

0

Ratio of fcpu/fsys is 2.

1

Ratio of fcpu/fsys is 1 i.e. fcpu = fsys.

9.1.8 SYS_CFG_KDIV macro

```
#define SYS_CFG_KDIV
```

Defined in: **sys_cfg.h**

User has to configure this with a value ranges from 1 to 16, used for both PLL and VCO bypass modes.

9.1.9 SYS_CFG_PDIV macro

```
#define SYS_CFG_PDIV
```

Defined in: **sys_cfg.h**

User has to configure this with a value ranges from 1 to 8, used for both PLL and VCO bypass modes.

9.1.10 SYS_CFG_NDIV macro

```
#define SYS_CFG_NDIV
```

Defined in: **sys_cfg.h**

User has to configure this with a value ranges from 1 to 128, used only for PLL mode.

9.1.10.1 Comments

Advisable value range is 20 to 100.

9.1.11 SYS_CFG_FIX_TC1130A_BUG macro

```
#define SYS_CFG_FIX_TC1130A_BUG
```

Defined in: **sys_cfg.h**

User can use this definition for software workaround done for TC1130A at system driver and not at module level.

1

Enbale software work-around for hardware bug fixes.

0

Disbale software work-around for hardware bug fixes.

9.2 Interrupt priorities configuration

The following priorities are used for interrupts. Corresponding to these priorities ISR code will be placed in Interrupt base Vector Table. The user can edit the priorities according to application requirements. These priorities will be static.

Priorities range from 1 to 255. Each interrupt should have a unique priority. 1 is the lowest priority and 255 is the highest priority.

9.3 GPIO Port Configuration Parameters

This section defines the configurable port settings of the peripherals. These macros define the following parameters:

Peripheral Module

- Name of the macro which includes the name of the peripheral and the port line (Transmit/Receive).

Port

- Port Number.

Pin

- Bit Number in the Port.

Dir

- Value of the bit in the Dir register.

Alt0

- Value of the bit in the Altsel0 register.

Alt1

- Value of the bit in the Altsel1 register.

Od

- Value of the bit in the Open Drain register.

Pullsel

- Value of the bit in the Pull up/Pull down selection register.

Pullen

- Value of the bit in the Pull up/Pull down enable register.

Note: User may use -1, to indicate an unused (or don't care) value.

These macros should be defined has a set of values in above sequence and separated by commas (,). E.g. #define SYS_GPIO_ASC0_TX 1, 7, 1, 1, -1, -1, -1, -1

10 Appendix C - The SYS HAL Appilcation Interface

10.1 TC1130 macro

```
#define TC1130
```

Defined in: **sys_api.h**

Controller type, this defines TC1130 to 1

10.2 SYS_PRODUCT macro

```
#define SYS_PRODUCT
```

Defined in: **sys_api.h**

Controller type, this define allows the user to select the type of the microcontroller.

10.2.1 SYS_STATUS

```
enum SYS_STATUS {
                                        SYS_GPIO_ERR,
                                        SYS_PRTY_ERR,
                                        SYS_CLK_ERR,
                                        SYS_SUCCESS
};
```

SYS HAL Status Enumeration

Defined in: **sys_api.h**

10.2.2 Members

SYS_GPIO_ERR
SYS_GPIO_ERR indicates a problem in the allocation of GPIO ports.

SYS_PRTY_ERR
SYS_PRTY_ERR indicates a problem in the allocation of interrupt priority.

SYS_CLK_ERR
SYS_CLK_ERR indicates a problem in the clock setting.

SYS_SUCCESS
SYS_SUCCESS indicates that an operation completed successfully.

10.2.3 Comments

The SYS_STATUS enum constants return the status from the SYS HAL functions.

10.3 SYS_TRANS_MODE

```
enum SYS_TRANS_MODE {
                                        SYS_TRNS_DMA,
                                        SYS_TRNS_PCP,
                                        SYS_TRNS_MCU_INT,
                                        SYS_TRNS_MCU
}SYS_TRANS_MODE;
```

Depending upon the system the HAL is operating in there may be several different options available regarding data transfers. Some systems have a DMA controller available, others have a PCP, some have both of these and some have neither. The system HAL provides constants which can be used to specify the desired data transfer option, the typedef name SYS_TRANS_MODE is used wherever such a constant is expected. The data transfer option must be initialised in the SSC_TRANSFER structure passed to the SSC_read and SSC_write API functions. If the transfer operation is not available in the system then SSC_ERR_NOT_SUPPORTED_HW will be returned.

Defined in: **sys_api.h**

10.3.1 Members

SYS_TRNS_DMA
Use the DMA controller to move the data

SYS_TRNS_PCP
Use the PCP to manage the transfer

SYS_TRNS_MCU_INT
Use the microcontroller unit to manage the transfer using interrupts.

SYS_TRNS_MCU
Use the microcontroller unit to manage the transfer by polling the SSC peripheral.

10.4 SYS HAL API Functions

10.4.1 SYS_gpio_alloc

SYS_STATUS SYS_gpio_alloc(IFX_SINT32 *SYS_port*, **IFX_SINT32** *SYS_pin*, **IFX_SINT32** *SYS_dir*, **IFX_SINT32** *SYS_altsel0*, **IFX_SINT32** *SYS_altsel1*, **IFX_SINT32** *SYS_od*, **IFX_SINT32** *SYS_pudsel*, **IFX_SINT32** *SYS_puden*)

GPIO Port allocation function, this function allocates the ports as defined by the user in the SYS CFG file. If the ports which the user wants to configure are already used by another peripheral, the function returns error, else if there are no conflicts, it returns SYS_SUCCESS

Defined in: SYS_API.H

10.4.1.1 Return Value

Returns SYS_STATUS

10.4.1.2 Parameters

SYS_port
GPIO Port number

SYS_pin
GPIO Port lines

SYS_dir
Value to be programmed in the GPIO Port Dir register

SYS_altsel0
Value to be programmed in the GPIO Port Altsel0 register

SYS_altsel1
Value to be programmed in the GPIO Port Altsel1 register

SYS_od
Value to be programmed in the GPIO Open drain control register

SYS_pudsel
Value to be programmed in the GPIO Pull-up/Pull-down select register

SYS_puden
Value to be programmed in the GPIO Pull-up/Pull-down enable register

- Check if the user settings are not conflicting with any other peripheral or not, if conflicting return error.

- If the register settings are legal, program the register settings in the GPIO else return an error.

10.4.2 SYS_gpio_free

SYS_STATUS SYS_gpio_free(IFX_SINT32 *SYS_port*, **IFX_SINT32** *SYS_pin*, **IFX_SINT32** *SYS_dir*, **IFX_SINT32** *SYS_altsel0*, **IFX_SINT32** *SYS_altsel1*, **IFX_SINT32** *SYS_od*, **IFX_SINT32** *SYS_pudsel*, **IFX_SINT32** *SYS_puden*)

GPIO Port free function, this function frees the ports which were allocated by the SYS_GPIO_alloc() function.

Defined in: SYS_API.H

10.4.2.1 Return Value

Returns SYS_STATUS

10.4.2.2 Parameters

SYS_port
GPIO Port number

SYS_pin
GPIO Port lines

SYS_dir
Ignored

SYS_altsel0
Ignored

SYS_altsel1
Ignored

SYS_od
Ignored

SYS_pudsel
Ignored

SYS_puden
Ignored

- This function frees the ports allocated by the SYS_gpio_alloc() function

10.4.3 SYS_gpio_get_port

IFX_SINT32 SYS_gpio_get_port(IFX_SINT32 *SYS_port*, **IFX_SINT32** *SYS_pin*, **IFX_SINT32** *SYS_dir*, **IFX_SINT32** *SYS_altsel0*, **IFX_SINT32** *SYS_altsel1*, **IFX_SINT32** *SYS_od*, **IFX_SINT32** *SYS_pudsel*, **IFX_SINT32** *SYS_puden*)

GPIO Port number return function, this function returns port number is being used.

Defined in: SYS_API.H

10.4.3.1 Return Value

Returns port number

10.4.3.2 Parameters

SYS_port
GPIO Port number

SYS_pin
GPIO Port lines

SYS_dir
Ignored

SYS_altsel0
Ignored

SYS_altsel1
Ignored

SYS_od
Ignored

SYS_pudsel
Ignored

SYS_puden
Ignored

10.4.4 SYS_gpio_get_pin_num

IFX_SINT32 SYS_gpio_get_pin_num(IFX_SINT32 *SYS_port*, **IFX_SINT32** *SYS_pin*, **IFX_SINT32** *SYS_dir*, **IFX_SINT32** *SYS_altsel0*, **IFX_SINT32** *SYS_altsel1*, **IFX_SINT32** *SYS_od*, **IFX_SINT32** *SYS_pudsel*, **IFX_SINT32** *SYS_puden*)

GPIO port line (pin number) return function, this function returns port line (pin number) is being used.

Defined in: SYS_API.H

10.4.4.1 Return Value

Returns port line (pin number).

10.4.4.2 Parameters

SYS_port
GPIO Port number

SYS_pin
GPIO Port lines

SYS_dir
Ignored

SYS_altsel0
Ignored

SYS_altsel1
Ignored

SYS_od
Ignored

SYS_pudsel
Ignored

SYS_puden
Ignored

10.4.5 SYS_clk_initialise

IFX_UINT32 SYS_clk_initialise(void)

Clock initialisation function, this function initialises the system clock by programming PLL_CLC register. For 20MHz external crystal oscillator it output 95MHz frequency, by using NDIV = 38, KDIV = 8 and PDIV = 1 values.

Defined in: SYS_API.H

11 Appendix D - Source file references

11.1 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*
usb_main.c	Implements the sample program to test USB HAL

11.2 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.

11.3 Other dependent files

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

*IIL – Implementation Independent Layer (non-hardware related)
*IDL – Implementation dependent Layer (hardware related)

12 APPENDIX E- Limitations & Bugs fixes

This section defines the Limitations of the USB LLD HAL.release version 4.2

12.1 Bugs fixed with current version

- Optimizations in handling the interrupts done, so that the code could work on SIS,NEC, Intel host controllers on all possible memory configuration with 48, 96, 144MHz
- Works with Automatic mode with TC1130 BB Step board
- Tasking toolchain: MMU alignment problem with Tasking 2.2r2 & Tasking 2.2r3 fixed, patches have been provided with this release.
- EP0 application provided which works both in transmit and receive

12.2 Limitations of the USB LLD Release

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.

5) Bulk mode limitation

In manual mode as well, if data reception rate is too fast(Bulk mode), the FIFO pointers gets currupted. For ex: in case of short packets, the packets arrive into FIFO within no time. These FIFO pointer corruptions are handled in software using manual mode(by invalidating the particular pipe until the process is corrupt). So it is recommended to execute the USB Code from internal RAM or SDRAM. If the code is executed from flash, it is recommended to sync the application timing so that FIFO corruptions are avoided.

12.3 Hardware Restrictions :

1) 2 kBytes buffer size

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.pdf

13 Quick User Reference for USB LLD

13.1 Installation of DAS Server and Hands-on DAS Demo Application:

Driver FileName: usbiov.inf (Copy the Thesycon folder from Blink_LED Folder to C:\) as indicated in below path.

Thesycon\USBIO_VL\V2.31\usbio\usbiov.inf

After connecting the USB peripheral to your PC, and power on for the first time, windows would report that new hardware has been detected and would you like to install the suitable driver. You would find the required ".inf" file inside the folder where DAS is installed on your PC(Search for a file named **"usbiov.inf"** inside Thesycon\USBIO_VL\V2.31\usbio folder).

Use the usb_main_das.c, application and build the USB LLD code and download the code onto hardware, and after execution, plug and play the USB cable, and after successful enumeration, windows finishes installing the driver, you may check that the four entries shown below are in the USB Section of Windows Device Manager (To open the Device Manager, right click on the "My Computer" icon on your desktop and click on "Manage". In the windows that pops up click on "Device Manager").

"Infineon - USB - IF0"

"Infineon - USB - IF1"

"Infineon - USB - IF2"

"Infineon - USB - IF3"

The USB Driver can also be found in the release package in the **USB_Driver** folder, update the USB driver to **"usbiov.inf"**

Run the application found in the "USB_V4.2\Demo_Application\Quick_LLD_Reference\TC1130_DAS_TEST_GUI.exe" folder, which pops up the following executable in the below mentioned diagram.

==========

Check for Manual Mode:

1. Buffer Count, Size: 2,64

2. 0Fill = not enabled X, 52, 170, 210, 500, 64, 128, 129, 256, 512

Check for Automatic Mode:

1. Buffer Count, Size: 2,512

2. 0Fill = enabled X, 52, 170, 210, 500, 982, 64, 128, 129, 256, 512, 2048

==========

1. Tick the Interface number and alternate setting number and add the EP number in decimal, for example, with the default, usb_main_das.c, EP1(129), EP2(2), have been configured on Interface 1(IF1) and Alternate Setting(AS1), and hence the configuration in the application are selected accordingly.

2. After selecting the configuration, click on **Do Button** which Register Device Notification, Create Event, builds the device list, and open the device and binds the in and out EP for communication. On successful enumeration and identification of the device, the following message would be dispayed:

RegisterDeviceNotification done

CreateEvent passed!

Building of device list passed!

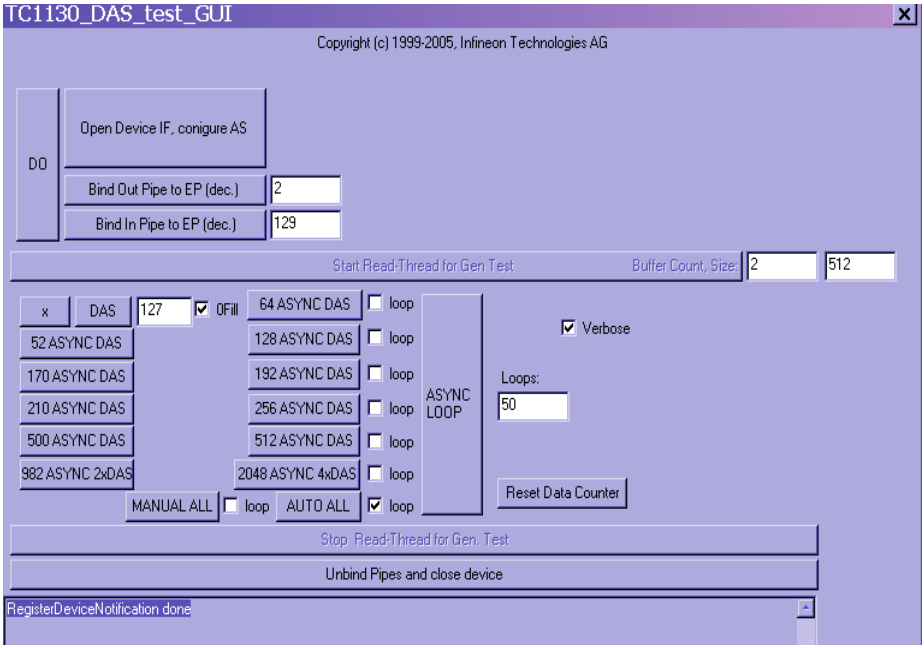
Device open Passed!

Binding to pipe for Out EP Passed!

Binding to pipe for In EP Passed!

or either click on Open Device IF, Configure AS, followed by Binding Out pipe to EP 2 and Bind IN pipe to EP 1 i.e, 129

Note for manual mode: Select Size of Buffer count to be 64, and also tick MANUAL ALL, whereas for AUTO mode, the same configuration as shown in below diagram can be used.



3. Select the Buffer count to 2, 512 for automatic test and 1,64 for manual mode test, [Please note the Limitation as mentioned in Release_Note_USB_LLD_v4_2.pdf with the release notes for selection of buffer count]

Default Buffer count is configured to 2, size to 512.

4. After selecting the Buffer count, **"Start read - Thread for Gen Test"**

5. Select the type of **"ASYNC DAS"** tranmission to be tested, and perform the asynchronous tranmission. Tick the box next to the tests, and Loops count and then click on Async loop to test the Async transmission in loop.

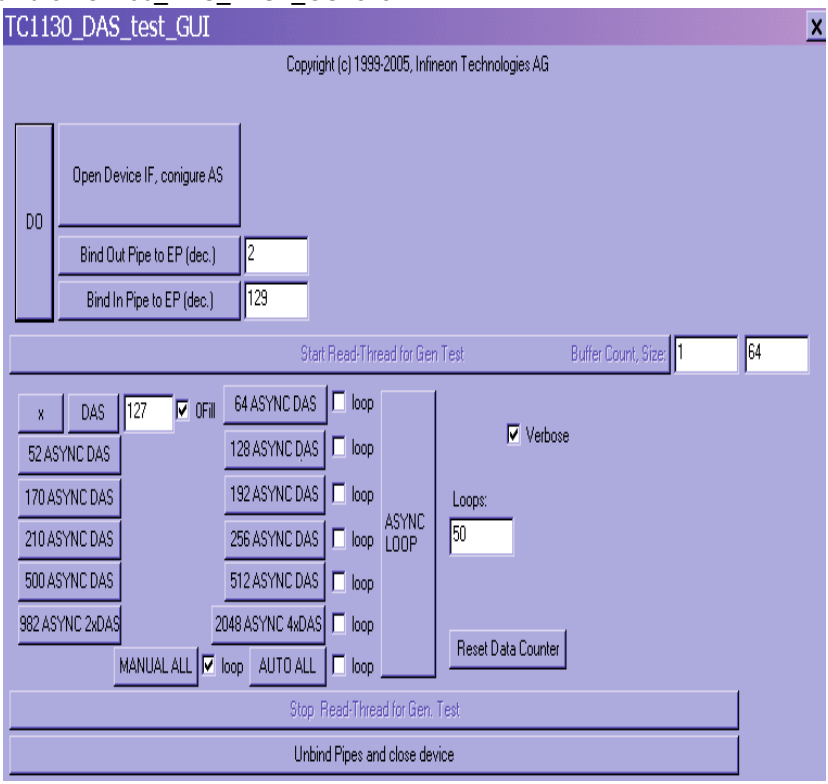
6. **"TC1130 Test flow"** tests all the asynchronous tests at one shot.

7. After the successful test **"Stop Read Thread for Gen Test"** and **"Unbind Pipes and Close Device".**

Manual mode Test

1. Select the **USB_MANUAL_MODE** in usb_idl_cfg.h file and recompile the project workspace.

2. Run the TC1130_DAS_TEST_GUI.exe



2. Tick the Interface number and alternate setting number and add the EP number in decimal, for example, with the default, usb_main_das.c, EP1(129), EP2(2), have been configured on Interface 1(IF1) and Alternate Setting(AS1), and hence the configuration in the application are selected accordingly.

3. After selecting the configuration, click on **Do Button** which Register Device Notification, Create Event, builds the device list, and open the device and binds the in and out EP for communication. On successful enumeration and identification of the device, the following message would be dispayed:

RegisterDeviceNotification done

CreateEvent passed!

Building of device list passed!

Device open Passed!

Binding to pipe for Out EP Passed!

Binding to pipe for In EP Passed!

or either click on Open Device IF, Configure AS, followed by Binding Out pipe to EP 2 and Bind IN pipe to EP 1 i.e, 129

4. Select the Buffer count to 2, 512 for automatic test and 1,64 for manual mode test, [Please note the Limitation as mentioned in Release_Note_USB_LLD_v4_2.pdf with the release notes for selection of buffer count]

Default Buffer count is configured to 2, size to 512.

5. After selecting the Buffer count, **"Start read - Thread for Gen Test"**

6. Select the type of **"ASYNC DAS"** tranmission to be tested, and perform the asynchronous tranmission. Tick the box next to the tests, and Loops count and then click on Async loop to test the Async transmission in loop.

7. **"TC1130 Test flow"** tests all the asynchronous tests at one shot.

7. After the successful test **"Stop Read Thread for Gen Test"** and **"Unbind Pipes and Close Device".**

s

GUI for a EP0 read check

This GUI has been developed to detect & decode a classor vendor request. To test the EP0 bi-directionality, we have to work with USBIO_CLASS_OR_VENDOR_REQUEST. (Page 98 on usbioman.pdf in the Thesycon directory)

1. Load the .elf present in the application folder through source navigator/ crossview respectively.
2. Run the application.
3. Plug and play the device.
4. Install the driver present in Driver folder usbiov.inf.
5. after successful enumeration, windows finishes installing the driver, you may check that the four entries shown below are in the USB Section of Windows Device Manager (To open the Device Manager, right click on the “My Computer” icon on your desktop and click on “Manage”. In the windows that pops up click on “Device Manager”).

"Infineon - USB - IF0"

"Infineon - USB - IF1"

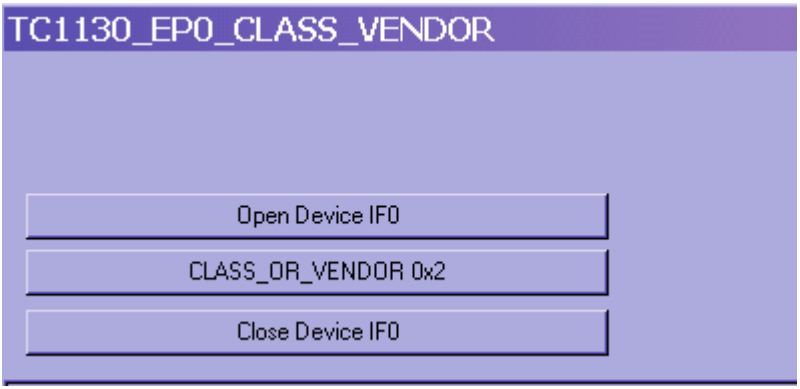
"Infineon - USB - IF2"

"Infineon - USB - IF3"

Run the application found in the “USB_V4.2\Demo_Application\Quick_LLD_Reference\TC1130_EP0_CLASS_VENDOR.exe”folder, which pops up the following executable in the below mentioned diagram.

6. Click on Open Device IF0, followed by CLASS_OR_VENDOR 0x2

and then click on close device IF0



On successful execution of application, the following result is obtained in the output window

```
CreateEvent passed!
Building of device list passed!
Device open passed!
Sending Vendor Request...
DATA UPLOAD passed!
10000000
01000000
11000000
00100000
10100000
01100000
11100000
00010000
: 0c0b0a09
: 100f0e0d
: 14131211
: 18171615
: 1c1b1a19
: 201f1e1d
: 24232221
: 28272625
: 2c2b2a29
: 302f2e2d
```

: 34333231
: 38373635
: 3c3b3a39
: 003f3e3d
Device and Pipes successfully closed!

http://www.infineon.com

Published by Infineon Technologies AG