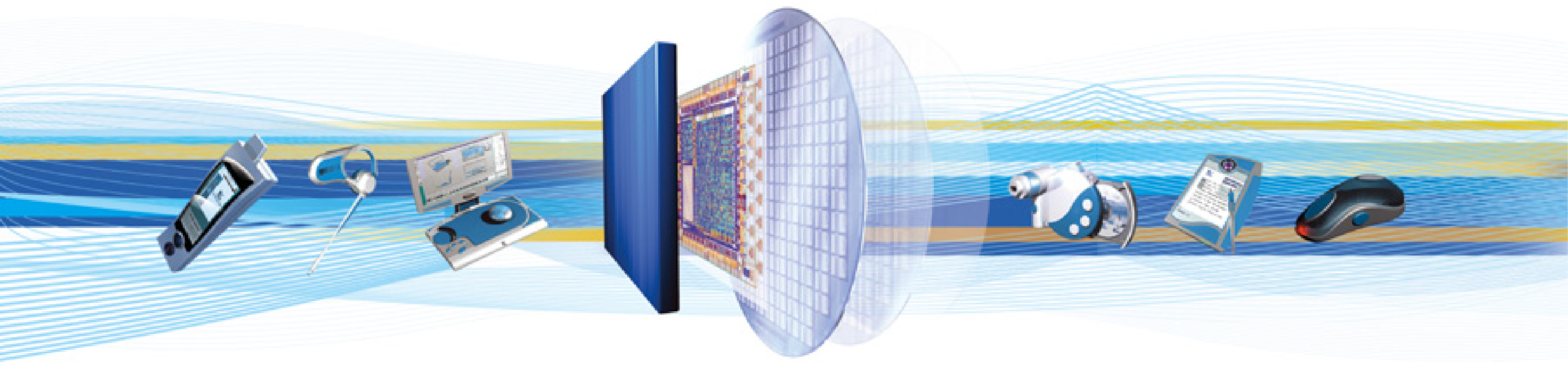
CSR Synergy®

CSR Synergy Framework 3.1.0

USB

API Description

August 2011



Cambridge Silicon Radio Limited

Churchill House
Cambridge Business Park
Cowley Road
Cambridge CB4 0WZ
United Kingdom

Registered in England and Wales 3665875

Tel: +44 (0)1223 692000
Fax: +44 (0)1223 692001
www.csr.com

Contents

Appendix

List of Tables

List of Figures

1 Introduction

1.1 Introduction and Scope

This document describes the requirements and functionality the CSR Synergy Framework mandates for communication via the USB interface.

It is assumed that the reader of this document has a basic understanding of the USB specification [USB-SPEC-2_0] and has read section “Volume Four – Part B – USB Transport Layer” of [BT-CORE-SPEC-2_1].

For detailed information about, firmware configuration options (PS Keys), the different power supply configurations, and some aspects of hardware design which commonly cause problems in relation to BlueCore and USB. It is recommended that the reader also refer [BT&USB-DESIGN-CON].

The function prototypes which is required for complying with the requirements and functionality described in this document is found in `csr_usb_com.h`

2 USB Interface

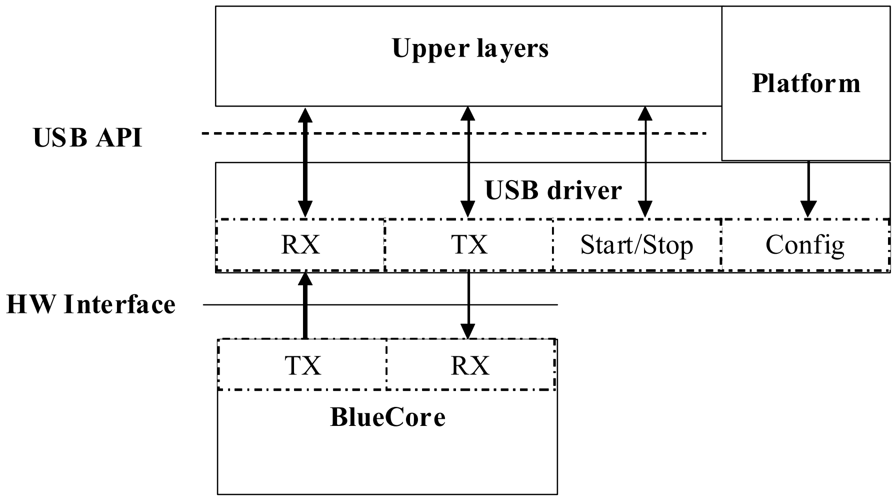


Figure 1: Illustration of where the USB API applies

2.1 Introduction

The USB API in the CSR Synergy Framework is designed as a simple send/receive interface which makes use of the HCI channels mapped to different kind of USB end points and different USB interfaces. The four different types of HCI traffic use all four USB transfer types across the two interfaces as follows:

- Interface 0
  - Control Endpoints: HCI commands
  - Interrupt Endpoint: HCI events
  - Bulk Endpoints: HCI ACL data
- Interface 1
  - Isochronous Endpoints: HCI SCO data

The HCI SCO interface implementation is however optional and should only be implemented if SCO over HCI is needed in the system.

Besides the required functionality, which is start, stop, rx and tx, many platforms support multiple USB ports and they therefore, need dynamic mapping to a specific device at startup (eg. you have two USB dongles connected and you want to be able to specify which to connect to when the device is powered up). Since this functionality is not needed on all platforms this functionality is not covered by the mandated part of the USB API. However for the purpose of testing and verification of BSPs provided with the CSR Synergy Framework, such an API has been implemented and is used by demo applications and this is described in Appendix A.

The CSR Synergy Framework is as described delivered with a set of BSPs, one for Windows-x86 (found in /bsp/ports/pcwin**)**, one for Linux-x86 (found in /bsp/ports/pclin**)** and one for Nucleus-ARM (found in /bsp/ports/bdb2**)**. Only the Windows-x86 BSP implements the USB API and it only support the channels on Interface 0 via a file descriptor driver. The actual Windows kernel driver source is not delivered with this BSP. Further, SCO over HCI is not support in the Windows-x86 BSP. Please refer to the Windows-x86 BSP (under pcwin/src/low_level/usb) for examples on how the USB API could be implemented on a platform with a file descriptor usb driver.

2.2 USB API description

2.2.1 CsrUsbDrvStart

Prototype

```
#include "csr_usb_com.h"

CsrBool CsrUsbDrvStart(CsrUint8 reset);
```

Description

This function is used for starting up the USB device. A call to this function happens before any data communication takes place with the USB driver. This is the first function called after system startup (where the driver is assumed to be in its default state) or directly after a call to `CsrUsbDrvStop()`, which happens in relation to restarting the USB communication.

It is also the responsibility of this function to register the background interrupt function associated with the receiving data path. This is done by means of a call to `CsrBgIntRegister()`, every time the USB driver has new data available for the upper layers it must set this. Please refer to section 2.2.4 for an example on how incoming data is delivered to upper layers.

Parameters

Type	Argument	Description
`CsrUint8`	`reset`	This parameter tells the driver whether this is the initial start or a subsequent start call. First time this function is called after system startup the value will be zero (this is before bootstrapping). And in any subsequent calls it will be higher than zero.

Table 1: Arguments to CsrUsbDrvStart

Returns

TRUE, if the USB device starts successfully. FALSE, if the USB device fails to start for any reason. In case of failure, the state of the USB driver must be the same as before this function call.

2.2.2 CsrUsbDrvStop

Prototype

```
#include "csr_usb_com.h"

CsrBool CsrUsbDrvStop(void);
```

Description

This function is called to stop (close) the USB device. It un-does all actions performed by `CsrUsbDrvStart()` e.g. free any allocated memory and unregister background interrupts.

Note: It is expected that the USB configuration does NOT change when this function is called

Parameters

None.

Returns

TRUE, if the USB device stops successfully. FALSE otherwise.

2.2.3 CsrUsbDrvTx

Prototype

```
#include "csr_usb_com.h"

CsrBool CsrUsbDrvTx(CsrUint8 channel, CsrUint8 *data, CsrUint32 size);
```

Description

This function is used for pushing data onto the USB TX channel buffers.

Note: The data pointer in this signal always carries complete HCI messages. It is the responsibility of the USB driver to send the complete HCI message as one USB transaction. A USB transaction may cover the transmission of several USB frames. Thus, the USB driver is responsible for, splitting the complete HCI message into smaller USB frames, in case the HCI message size is larger than the allowed max allowed USB frame size.

Parameters

Type	Argument	Description
`CsrUint8`	`channel`	Can be one of the three following values: TRANSPORT_CHANNEL_HCI -> means data must be sent via the USB control endpoint on interface 0 TRANSPORT_CHANNEL_ACL -> means data must be sent via the USB bulk endpoint on interface 0 TRANSPORT_CHANNEL_SCO -> means data must be sent via the USB isochronous endpoint on interface 1 **Note:**
`CsrUint8 *`	`data`	Pointer to the data payload which must be sent
`CsrUint32`	`size`	Number of bytes in data pointer which should be written to the USB TX channel.

Table 2: Arguments to CsrUsbDrvTx

Returns

If the USB TX buffer cannot hold an extra `len` bytes when this function is called it must return FALSE. If all the bytes are pushed to the TX buffer the function must return TRUE.

2.2.4 How to send incoming data to upper layers

Incoming data from the USB driver needs to be passed to the upper layers by means of background interrupts in order for data to be handled in scheduler context. An example of how this can be achieved is described below.

In the example a function named `CsrUsbDrvRx()` of the type `CsrBgintHandler` is registered as background interrupt function. The background interrupt which is stored in the variable `usb_bgint_rx`.

```
    usb_bgint_rx = CsrBgintRegister(CsrUsbDrvRx, NULL, "UsbDrv_Rx");
```

Please refer to section 9 of [SYN-FW-COAL-API] for more information about the background interrupt API. Every time the background interrupt is set the scheduler will make sure to call CsrUsbDrvRx() function which must then fill in the incoming data into a `MessageStructure` found in csr_transport.h and pass it on to higher layers via `CsrTmBlueCoreUsbDeliverMsg()`. a pseudo like example below.

```
Void CsrUsbDrvRx(void *unused)
{
    MessageStructure hciMsg;
    hciMsg.buf = incomingData; /* must be a CsrPmalloc'ed pointer */
    hciMsg.buflen = length;

    if(dataCameFromInterruptEndpoint)
        hciMsg.chan = TRANSPORT_CHANNEL_HCI;
    else if(dataCameFromBulkEndpoint)
        hciMsg.chan = TRANSPORT_CHANNEL_ACL;
    else if(dataCameFromIsochroneusEndpoint)
        hciMsg.chan = TRANSPORT_CHANNEL_SCO;

    /* pass it to higher layers */
    CsrTmBlueCoreUsbDeliverMsg(&hciMsg, 0);

    if(moreDataWaiting)
        CsrBgintSet(usb_bgint_rx);
}
```

Note: It is the responsibility of the USB driver to reassemble the USB fragments (USB frames) so only complete USB transactions (i.e. complete HCI messages) are passed to upper layers.

Appendix A API for configuring the USB device

As mentioned earlier selecting/configuring the USB device is not a requirement for all platforms and hence it is not a requirement the CSR Synergy Framework mandates the platform to implement. For the purpose of testing and verification, the BSPs provided with the CSR Synergy Framework have implemented an API, which is used by the demo applications and which can serve as an inspiration for how such an API could be designed on other platforms.

The prototype for this API can be found in `csr_usb_init.h`. This API is named `CsrUsbDrvConfigure()` and is defined as.

An example of how this API is used is illustrated in Figure 2.

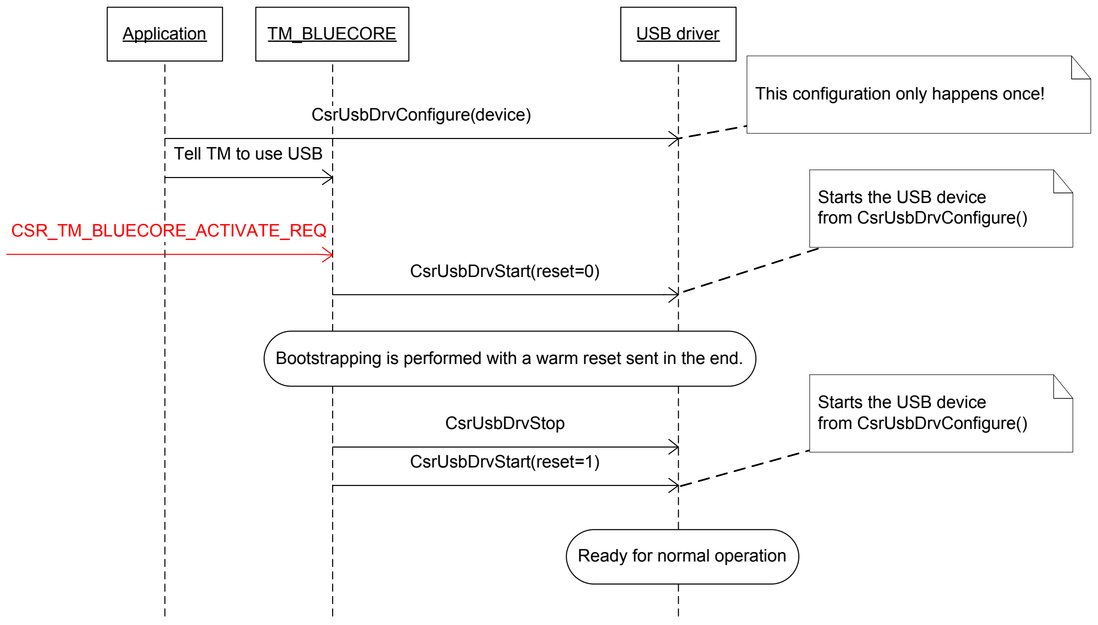

Application
TM_BLUECORE
USB driver
This configuration only happens once!
CsrUsbDrvConfigure(device)
Tell TM to use USB
Starts the USB device from CsrUsbDrvConfigure()
CSR_TM_BLUECORE_ACTIVATE_REQ
CsrUsbDrvStart(reset=0)
Bootstrapping is performed with a warm reset sent in the end.
Starts the USB device from CsrUsbDrvConfigure()
CsrUsbDrvStop
CsrUsbDrvStart(reset=1)
Ready for normal operation

Figure 2: Example of the usage of CsrUsbDrvConfigure

Prototype

```
#include "csr_usb_init.h"

void CsrUsbDrvConfigure(char *device);
```

Description

This function is used for telling the USB driver which USB device to connect.

Parameters

Type	Argument	Description
`char *`	`device`	The name of the device eg. `//./csr1`

Table 3: Arguments to CsrUsbDrvConfigure

Returns

Nothing.

3 Document References

Ref	Title

Terms and Definitions

[SYN-FRW-COAL-API]	CSR Synergy Framework COAL API. Doc. Api-0001-coal
[BT&USB-DESIGN-CON]	**Bluetooth and USB Design Considerations (CS-101412-ANP)**
[USB-SPEC-2_0]	USB specification 2.0
[BT-CORE-SPEC-2_1]	Bluetooth Core Specification 2.1
USB	Universal Serial Bus
BC	BlueCore
BlueCore®	Group term for CSR's range of Bluetooth wireless technology chips
CSR	Cambridge Silicon Radio

Document History

Revision	Date	History
1	30 NOV 09	Ready for release 2.0.0
2	20 APR 10	Ready for release 2.1.0
3	OCT 10	Ready for release 2.2.0
4	NOV 10	Ready for release 3.0.0
5	Aug 11	Ready for release 3.1.0

TradeMarks, Patents and Licences

Life Support Policy and Use in Safety-critical Compliance

Performance and Conformance

Refer to www.csrsupport.com for compliance and conformance to standards information.