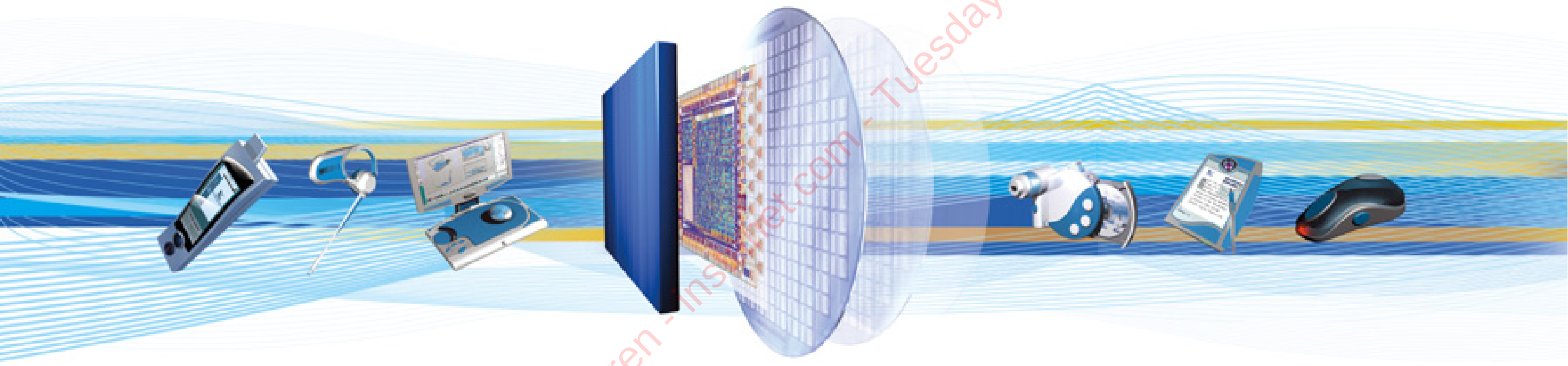
BlueCore®

Bluetooth and USB Design Considerations

Application Note

Issue 4



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

Document History

Revision	Date	History
bcore-an-069Pa	10 MAY 05	Original publication of this document.
CS-101412-ANP2	29 SEP 08	Updated with specific BlueCore5 information, battery charging section and changes to reflect specification ECNs.
3	04 DEC 08	Corrected figure notation in Figures 2.1, 2.2, 2.3 and 2.4 plus editorial.
4	11 JAN 10	Corrected PS Key names. Added PS Keys. Included 3rd method of booting BlueCore into USB. Editorial corrections.

TradeMarks, Patents and Licences

Contents

List of Figures

List of Tables

1 Introduction

One of the host interfaces available on **BlueCore**® is a full-speed (12Mbits/s) USB interface.

When correctly integrated, the silicon is compliant with the *Universal Serial Bus Specificati*on, Revision v2.0 (USB v2.0 Specification), which is available from http://www.usb.org. CSR strongly recommends that designers of circuits using BlueCore's USB interface read this specification. It contains valuable information on aspects such as PCB track impedance, supply inrush current, product labelling and USB certification. This document extensively references the specification.

USB is one of the original three HCI transports defined in the first version of the Bluetooth® wireless technology specification; in section H:2 of the v1.0 specification. (Section H:1 contains the core HCI specification, while sections H:3 and H:4 define RS232 and simple UART transports respectively.) The Bluetooth specification defines how HCI traffic flows over the USB interface.

This document initially summarises the way the HCI protocol flows over USB and the parts of the USB specification that most affect Bluetooth devices, in particular explaining the different power supply configurations. It then describes some aspects of hardware design that commonly cause problems.

The last sections of this document describe system issues: USB Suspend, USB Selective Suspend and Wake on Bluetooth.

At all stages, this document describes relevant firmware configuration options, known as Persistent Store Keys (PS Keys). An appendix summarises all the USB Persistent Store Keys.

This document assumes some technical knowledge of Bluetooth and BlueCore (for example the use of the HCI interface and PS Keys) and some familiarity with USB.

2 USB Basics

This section summarises some pertinent aspects of USB that impact Bluetooth operation. It is background information for the subsequent chapters.

There are three ways to enable BlueCore's USB interface:

- The most common way is to set PSKEY_HOST_INTERFACE to 2.
- It is also possible to set PSKEY_HOST_INTERFACE_PIO_USB so that if a particular PIO line is high at boot time the host interface is set to USB, and if the PIO line is low then the value setting in PSKEY_HOST_INTERFACE is used (e.g. BCSP).
- The third method of enabling the USB interface also uses PIO lines to select the host interface at boot. The exact PIO lines vary between BlueCore generations. See the *Booting BlueCore ROM* document for more details.

PS Key Name	Location	Default	Setting	Description
PSKEY_HOST_INTERFACE	0x01f9	1	2	Change from default of 1 (BCSP) to 2 (USB) to enable USB interface.
PSKEY_HOST_INTERFACE_PIO_USB	0x0250	–	0 - 15	Set to a value between 0 and 15 to force the use of the USB interface when that PIO line pulled high, overriding PSKEY_HOST_INTERFACE.

Table 2.1: PS Keys to Enable USB interface

2.1 Architecture

The USB physical interconnect follows a tiered star topology. The Host system (e.g. the PC) resides on Tier 1. There is only ever one Host system, and its Root Hub is the only connection point on Tier 1. Devices that connect to the Root Hub's downstream ports are on Tier 2. There are two types of device: Hub devices and Function devices. Hub devices have a single upstream port to connect to the Hub on the tier above and two or more downstream ports to enable connections to devices on the next tier down; they do not provide any functionality other than fanning out a single downstream USB port into multiple downstream ports. Function devices provide other functionality; they have a single upstream port. Timing considerations limit the maximum number of tiers to seven. (See the *USB v2.0 Specification*, section 4.1.1, page 16 for more details.)

BlueCore is always a Function device, connected to either a Root or device Hub.

Note:

A direct connection to the Root Hub is often preferable if the system must support all options for Wake on Bluetooth. See section 4, USB Suspend and Bluetooth Low Power Modes, for more details.)

2.2 Power Distribution and Suspend Modes

From the USB v2.0 Specification, section 4.3.1, page 18:

> Each USB segment provides a limited amount of power over the cable. The host supplies power for use by USB devices that are directly connected. In addition, any USB device may have its own power supply. USB devices that rely totally on power from the cable are called *bus-powered* devices. In contrast, those that have an alternate source of power are called *self-powered* devices. A hub also supplies power for its connected USB devices.

Bus-powered devices can either be low-power or high-power: less than 1 unit load or between 1 and 5 unit loads respectively, where one USB unit load is 100mA (at the 5V nominal VBUS voltage). This gives three power classes (from the *USB v2.0 Specification*, section 7.2.1, page 199, but replacing "unit loads" with the corresponding currents):

Low-power bus-powered functions: All power to these devices comes from VBUS. They may draw no more than 100mA at any time.

High-power bus-powered functions: All power to these devices comes from VBUS. They must draw no more than 100mA on power-up and may draw up to 500mA after being configured.

Self-powered functions: may draw up to 100mA from VBUS to allow the USB interface to function when the remainder of the function is powered down. All other power comes from an external (to the USB) source.

The USB specification describes how devices can be placed into a low-power state (from the *USB v2.0 Specification*, section 7.1.7.6, page 154):

All devices must support the Suspend state. Devices can go into the Suspend state from any powered state. They begin the transition to the Suspend state after they see a constant Idle state on their upstream facing bus lines for more than 3.0ms. The device must actually be suspended, drawing only suspend current from the bus after no more than 10ms of bus inactivity on all its ports. Any bus activity on the upstream facing port will keep a device out of the Suspend state.

…

While in the Suspend state, a device must continue to provide power to its D+ (full-/high-speed) or D- (lowspeed) pull-up resistor to maintain an idle so that the upstream hub can maintain the correct connectivity status for the device.

Because BlueCore is a full-speed device, it is required to maintain power to a D+ pull-up resistor. The presence of this pull-up voltage allows the upstream hub to detect the presence of the device. Section 0,

USB Enumeration, describes the function of the pull-up resistor and its importance in the USB enumeration process.

From a system-level perspective, there are two types of Suspend: Global and Selective. From the *USB v2.0 Specification*, sections 7.1.7.6.1 and 7.1.7.6.2, page 155:

Global suspend is used when no communication is desired anywhere on the bus and the entire bus (from the Root Hub down) is placed in the Suspend state.

Segments of the bus can be selectively suspended by sending the command SetPortFeature(PORT_SUSPEND) to the hub port to which that segment is attached. The suspended port will block activity to the suspended bus segment, and devices on that segment will go into the Suspend state after the appropriate delay as described above.

Note:

From the device perspective, there is no difference between the two types of suspend: devices see the same signalling, “a constant Idle state on their upstream facing bus lines for more than 3.0ms”. Therefore, by definition, BlueCore supports both Global and Selective Suspend, as does any USB certified device. The ability for a system to support Selective Suspend depends solely on the capabilities of the Host.

While suspended, a USB device must obey strict limits on the amount of current drawn from USB VBUS. Up until April 9th 2008 the suspend currents were limited to 500uA. After that date an ECN to the specification allows 2.5mA current draw in suspend mode in all configurations. From the *USB v2.0 Specification ECN*, section 7.2.3:

All USB Devices (except bus powered hubs) may draw up to 2.5mA during suspend. … When computing suspend current, the current from VBUS through the bus pull-up and pull-down resistors must be included.

Self-powered devices are limited in the same way as bus-powered devices, but having access to a power source other than USB VBUS means that they can continue to draw more than 2.5mA from this alternative source if the (non-USB) systems design allows for it.

The pull-up resistor at the device is 1.5 kΩ (nominal). The pull-down resistor at the hub is 14.25kΩ to 24.80kΩ. The pull-up voltage is nominally 3.3V, which means that holding one of the signal lines high takes approximately 200µA, leaving only 2.3mA available from a 2.5mA budget.

Devices exit from Suspend using the Resume procedure. From the *USB v2.0 Specification*, section 7.1.7.7, page 156:

If a device is in the Suspend state, its operation is resumed when any non-idle signalling is received on its upstream facing port. Additionally, the device can signal the system to resume operation if its remote wakeup capability has been enabled by the USB System Software.

Note:

For a device to initiate Resume via Remote Wakeup it must both support Remote Wakeup and have the feature enabled by the USB System Software. BlueCore, when properly configured, supports Remote Wakeup, but if the Host software does not, then the feature is not used.

All of these requirements and restrictions can influence Bluetooth operation. Section 4, USB Suspend and Bluetooth Low Power Modes, describes the relationships in detail.

Some of these restrictions can be relaxed for USB devices that do not require USB certification (e.g. internal USB modules). Section 2.4 describes some of the options that this opens up.

2.2.1 Low-power Bus-powered BlueCore Device

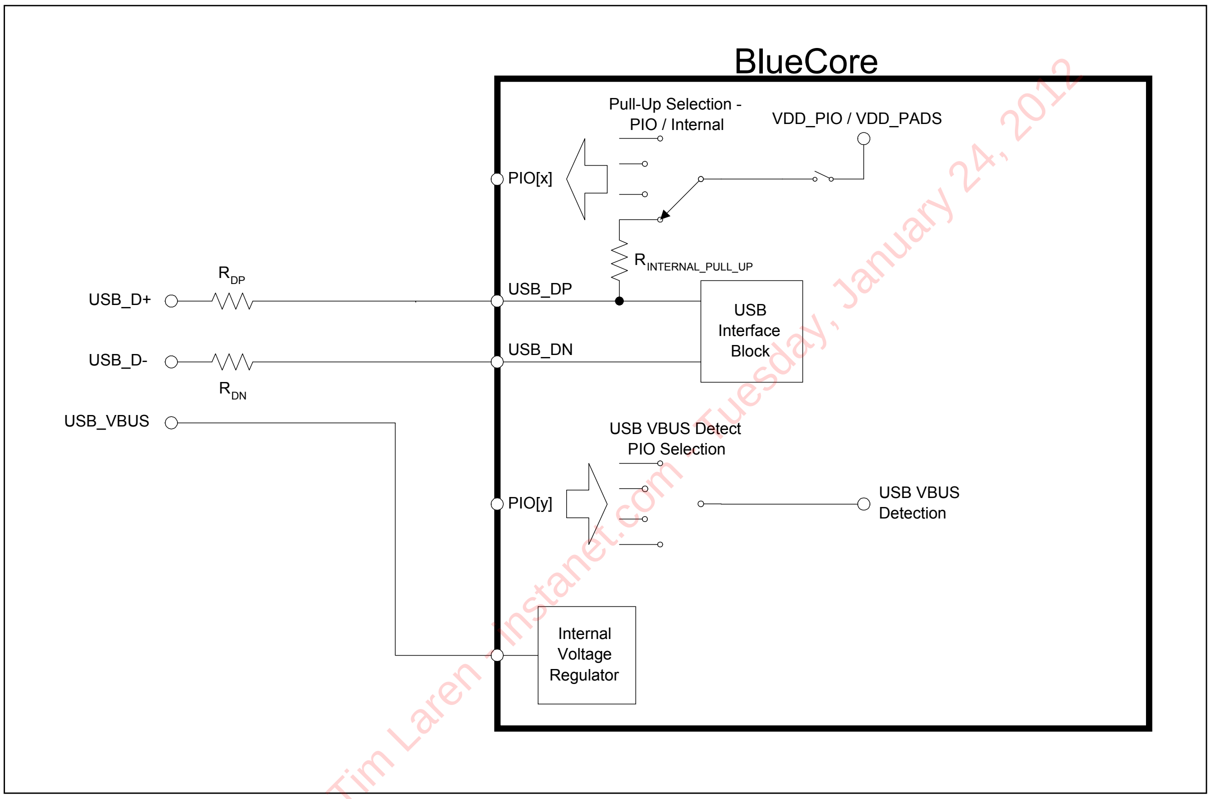


Figure 2.1: Low-power Bus-powered BlueCore Device Configuration

In this configuration, BlueCore never draws more than 100mA, so when used on its own in a bus-powered configuration it is always low-power. PSKEY_USB_MAX_POWER holds the current draw reported by BlueCore during device enumeration. The key's value defaults to zero. There is no need to change this value for a low-power device since the upstream Hub allocates all USB devices a minimum 100mA of current draw capacity.

USB VBUS detection is required on self-powered devices to determine whether the upstream Hub is active: if it is inactive then no voltage is applied to the pull-up resistor. In a bus-powered device the pull-up voltage itself is derived from USB VBUS, so such a check is redundant. USB VBUS detection is configured via PSKEY_USB_PIO_VBUS. The key defaults to *Not Present*, which disables the detection check: the firmware assumes that USB VBUS is always present. For bus-powered devices, there is no need to set this key (and if it is present it should be deleted).

For bus-powered devices the internal D+ pull-up resistor can be used. Configure pull-up selection using PSKEY_USB_PIO_PULLUP. The key's value defaults to 16, which is a *magic number* that enables the internal pull-up. For bus-powered devices, there is no need to change this setting. (Setting it to a value between 0 and 15 raises that PIO line high in order to drive an external pull-up resistor.)

Note:

Although Figure 2.1 shows the use of an internal voltage regulator, some variants of BlueCore do not contain this function block and an external regulator is required. However, this does not change any other aspect of the device configuration.

PS Key Name	Location	Default	Setting	Description
PSKEY_USB_MAX_POWER	0x02c6	0	0	Maximum current draw of device in units of 2mA. Keep at default of 0 for low-power devices (where max current draw is <100mA).
PSKEY_USB_PIO_VBUS	0x02d1	–	–	Values between 0 and 15 indicate PIO line to use to monitor USB VBUS. If key is not present then firmware assumes that USB VBUS is always present. Keep at default (not present) for bus-powered devices.
PSKEY_USB_ATTRIBUTES_POWER	0x03f2	0x0001	0x0000	A presentation key for bit 7 of USB Attributes bitmap (field bmAttributes). Bit 7 maps to self-powered. Changes to this key are reflected in bit 7 of PSKEY_USB_ATTRIBUTES (location 0x025c) and visa versa. Change to zero, for bus-powered devices.
PSKEY_USB_PIO_PULLUP	0x02d0	16	16	Values between 0 and 15 indicate PIO line to use to enable and disable USB D+/D- pull-up resistor. If key is not present then the firmware will not use any PIO line. The value of 16 is a *magic number* that enables the use of an internal pull-up on the USB D+ line. Keep at default of 16 for bus powered devices.

Table 2.2: PS Keys for Low-power Bus-powered BlueCore Device

2.2.2 High-power Bus-powered BlueCore Device

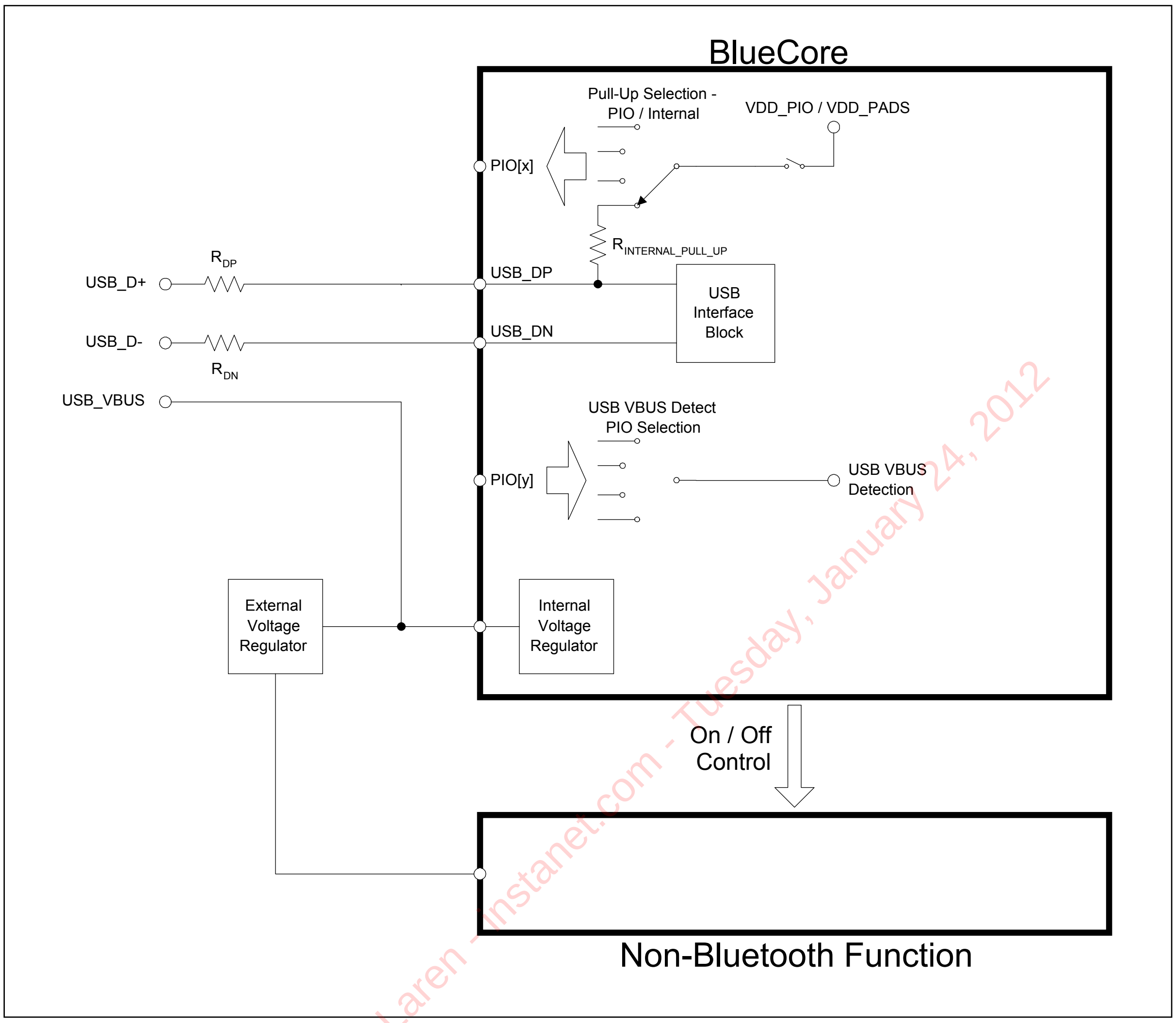


Figure 2.2: High-power Bus-powered BlueCore Device Configuration

In some cases, BlueCore may be combined with additional functions (such as a battery charger) that also draw power from USB VBUS. The device is classed as high-power if the total maximum current draw rises above 100mA.

BlueCore must report the high-power current requirement during enumeration. PSKEY_USB_MAX_POWER holds this information in the same format as the USB bMaxPower field. From the *USB v2.0 Specification*, section 9.6.3, page 266:

> Maximum power consumption of the USB device from the bus in this specific configuration when the device is fully operational. Expressed in 2mA units (i.e., 50 = 100mA).

Note the unusual “2mA units” requirement.

When attached, the upstream may or may not be able to supply the necessary current for full functionality. BlueCore must therefore have the ability to enable and disable the non-Bluetooth function depending on whether the request for high-power is granted or not. A VM application is typically used to enable and disable the non-Bluetooth function by toggling a PIO line.

Apart from the additional power configuration requirements, the PS Key configuration is the same as for a low-power bus-powered device.

Note:

Although Figure 2.2 shows the use of an internal voltage regulator, some variants of BlueCore do not contain this function block and power may be taken from the same external regulator as the non-Bluetooth function. However, this does not change any other aspect of the device configuration.

PS Key Name	Location	Default	Setting	Description
PSKEY_USB_MAX_POWER	0x02c6	0	X	Maximum current draw of device in units of 2mA. Set to X, where X is the maximum power in 2mA units for high-power devices (where max current draw is >100mA).
PSKEY_USB_PIO_VBUS	0x02d1	–	–	Values between 0 and 15 indicate PIO line to use to monitor USB VBUS. If key is not present then firmware assumes that USB VBUS is always present. Keep at default (not present) for bus-powered devices.
PSKEY_USB_ATTRIBUTES_POWER	0x03f2	0x0001	0x0000	A presentation key for bit 7 of USB Attributes bitmap (field bmAttributes). Bit 7 maps to self-powered. Changes to this key are reflected in bit 7 of PSKEY_USB_ATTRIBUTES (location 0x025c) and visa versa. Change to zero, for bus-powered devices.
PSKEY_USB_PIO_PULLUP	0x02d0	16	16	Values between 0 and 15 indicate PIO line to use to enable and disable USB D+/D- pull-up resistor. If key is not present then the firmware will not use any PIO line. The value of 16 is a *magic number* that enables the use of an internal pull-up on the USB D+ line. Keep at default of 16 for bus powered devices.

Table 2.3: PS Keys for High-power Bus-powered BlueCore Device

2.2.3 Self-powered BlueCore Device

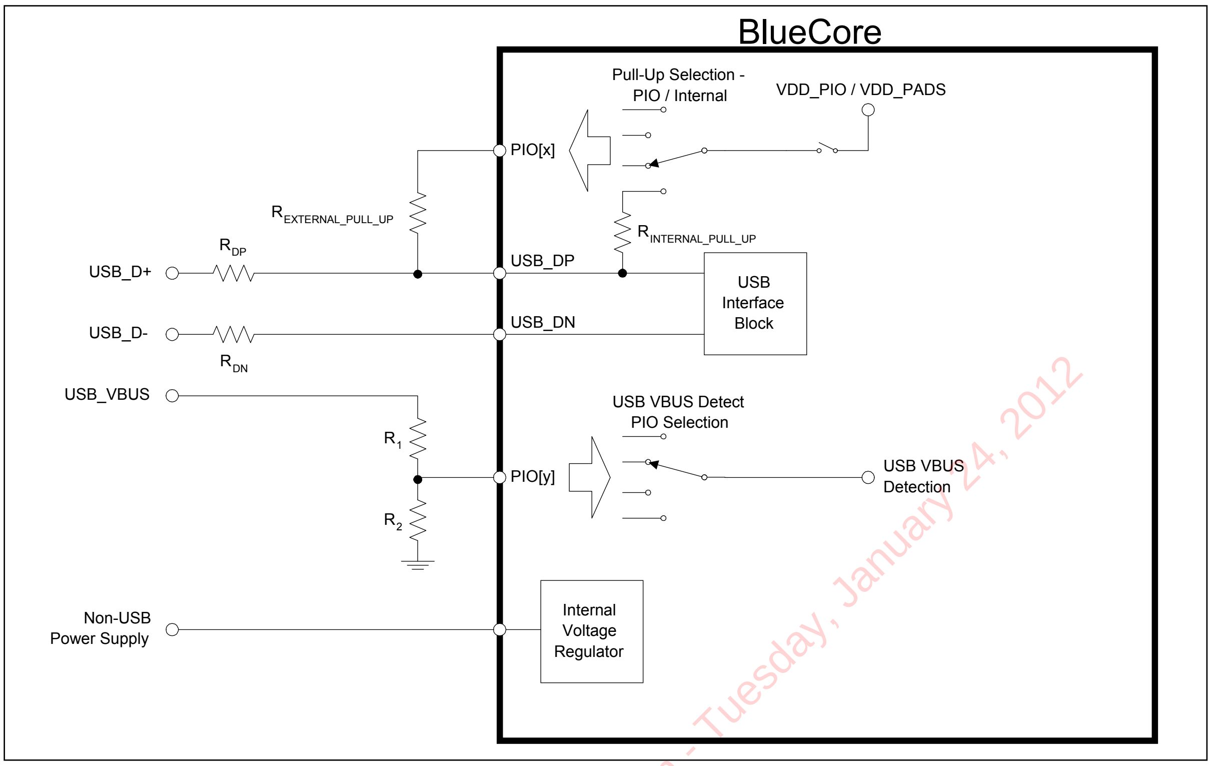


Figure 2.3: Self-powered BlueCore Device Configuration (BlueCore1-BlueCore4 and BlueCore6 Onwards)

g

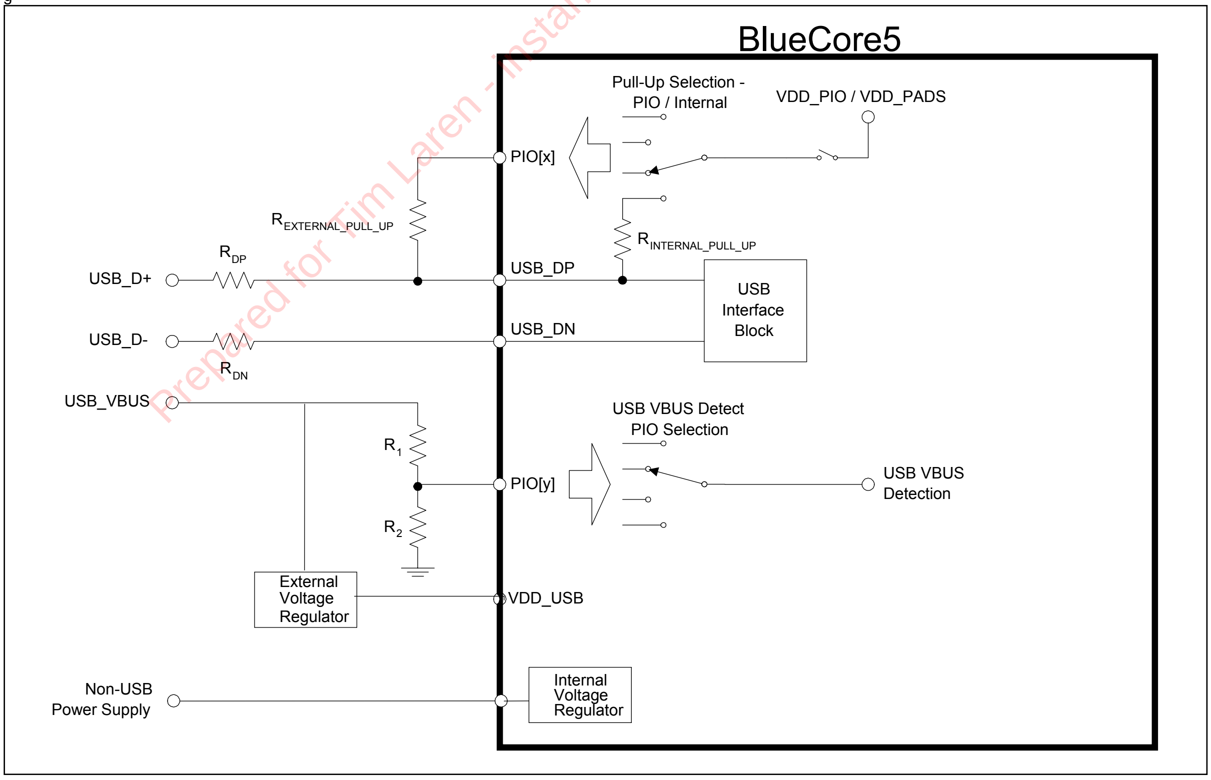


Figure 2.4: Self-powered BlueCore Device Configuration (BlueCore 5-FM, BlueCore 5-Audio ROM, BlueCore 5-Multimedia and MC601/MC603)

When running from an alternative power source, BlueCore's current draw from USB VBUS is a lot less than 1mA and certainly never rises above 100mA, so the reported current draw at enumeration can be the same as for a low-power device. Therefore, PSKEY_USB_MAX_POWER can be left at its default value of zero.

USB VBUS detection is required on self-powered devices in order to determine whether the upstream Hub is active: if it is inactive then no voltage is applied to the pull-up resistor. In a bus-powered device the pull-up voltage itself is derived from USB VBUS, so such a check is redundant. Configure USB VBUS detection using PSKEY_USB_PIO_VBUS. The key defaults to *Not Present*, which disables the detection check: the firmware assumes that USB VBUS is always present. For Self-powered devices, this key should be set to a value corresponding to the PIO line used to detect the presence of USB VBUS. A potential divider is required to step down the USB VBUS input voltage to a level suitable for the PIO line chosen (typically 3.3V, although 1.8V is also possible depending on the exact PIO line used and the configuration of the pad power supplies (see the individual BlueCore variant datasheets for more information).

For bus-powered devices, the internal D+ pull-up resistor can be used. However, for self-powered devices there is a problem: the device may be connected to the upstream Hub while the Hub is active but the device lacks its external (non-USB) power supply. This exposes BlueCore to a voltage on the USB VBUS detection input while un-powered and puts the behaviour of the internal pull-up resistor in an unknown state. The use of an external pull-up resistor is therefore strongly recommended. Configure pull-up selection using PSKEY_USB_PIO_PULLUP. The key's value defaults to 16, which is a *magic number* that enables the internal pull-up. For self-powered devices, there this key should be set to a value between 0 and 15, which causes the corresponding PIO line to go high in order to drive an the external pull-up resistor.

Special Considerations for BlueCore5-FM, BlueCore5-Multimedia, BlueCore5-Audio ROM and MC601/MC603 Devices in Self-powered Configuration

These devices have a leakage path that in some circumstances can cause USB_DP to rise above the maximum 400mV required by section 7.2.1 of the *USB v2.0 Specification* when the USB VBUS supply is removed. To ensure compliance with this test the VDD_USB pin must be supplied via an external 3.3V regulator powered by the USB VBUS. The VDD_USB pin also supplies the UART pins, so it should be noted that the supply to these pins will be affected when the USB VBUS supply is removed.

Note:

Although Figure 2.3 shows the use of an internal voltage regulator, some variants of BlueCore do not contain this function block and an external regulator is required. However, this does not change any other aspect of the device configuration.

PS Key Name	Location	Default	Setting	Description
PSKEY_USB_MAX_POWER	0x02c6	0	0	Maximum current draw of device in units of 2mA. Keep at default of 0 for self-powered devices.
PSKEY_USB_PIO_VBUS	0x02d1	–	Y	Values between 0 and 15 indicate PIO line to use to monitor USB VBUS. If key is not present then firmware assumes that USB VBUS is always present. Set to Y, where Y is the PIO line connected to the VBUS monitoring circuit for self-powered devices.
PSKEY_USB_ATTRIBUTES_POWER	0x03f2	0x0001	0x0000	A presentation key for bit 7 of USB Attributes bitmap (field bmAttributes). Bit 7 maps to self-powered. Changes to this key are reflected in bit 7 of PSKEY_USB_ATTRIBUTES (location 0x025c) and visa versa. Leave at default of 1 for self-powered devices.
PSKEY_USB_PIO_PULLUP	0x02d0	16	X	Values between 0 and 15 indicate PIO line to use to enable and disable USB D+/D- pull-up resistor. If key is not present then the firmware will not use any PIO line. The value of 16 is a *magic number* that enables the use of an internal pull-up on the USB D+ line. Set to X, where X is the number of the PIO line connected to the D+ pull-up resistor, for self-powered devices.

Table 2.4: PS Keys for Self-powered BlueCore Device

2.3 USB Enumeration

When a USB device is attached or removed, the host uses a process known as bus enumeration to identify and manage the necessary device state changes. Full details of the process are in section 9.1.2, page 243 of the *USB v2.0 Specification*, but the sequence can be summarised as follows:

1. The Hub detects attachment of the new device. The pull-up resistor at the device (on USB D+ for a full-speed device like BlueCore, on USB D- for a low-speed device) is 1.5 kΩ (nominal). The pull-down resistor at the hub is 14.25kΩ to 24.80kΩ. The voltage on the appropriate connection at the Hub therefore rises. It is this voltage high that enables the Hub to detect the attachment.
2. The Hub reports its change of state to the Host.
3. The Host queries the Hub to discover the nature of the change.
4. The Host enables the downstream port on the hub that the new device is attached to as well as a USB Reset to ensure the device's USB interface is in a known state.
5. The Host assigns the device a unique address (all USB devices initially connect on address zero) and reads its configuration information.

The hub both initially identifies the attachment of the device and determines its continued presence through the pulling-up of the D+ line, which effectively determines the idle state for that section of the bus: D+ high and D- low. If the idle state ever changes to both D+ and D- low, then that indicates the disconnection of the device.

(Low-speed devices pull the D- line high, which reverses the polarity of the idle, J and K states. See *USB v2.0 Specification*, section 7.1.7.1, table 7-2, page 145 for details.)

The configuration information is contained in USB Descriptors. Section 9.6 of the *USB v2.0 Specification* describes the standard USB device descriptors. Much of this information is fixed, but several fields are adjustable via PS Keys.

The default USB Descriptors define the two USB interfaces that are required for Bluetooth operation. A USB device can have multiple logical interfaces each of which can contain multiple endpoints. From the *USB v2.0 Specification*, section 5.3.1, page 33:

> An endpoint is a uniquely identifiable portion of a USB device that is the terminus of a communication flow between the host and device.

Each interface must have a Control endpoint. There are four classes of endpoint. Each maps directly to one of the four types of data transfer:

- Control Transfer: Supports configuration/command/status type communication flows between client software and its function.
- Isochronous Transfer: Provides the following:
  - Guaranteed access to USB bandwidth with bounded latency
  - Guaranteed constant data rate through the pipe as long as data is provided to the pipe
  - In the case of a delivery failure due to error, no retrying of the attempt to deliver the data
- Interrupt Transfer: Supports devices that need to send or receive data infrequently but with bounded service periods.
- Bulk Transfer: Supports devices that need to communicate relatively large amounts of data at highly variable times where the transfer can use any available bandwidth.

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
  - Control Endpoints: N/A (USB control traffic only)
  - Isochronous Endpoints: HCI SCO Data

Note:

A USB device has only one pair of control endpoints that are shared between all interfaces.

When a Device Firmware Upgrade (DFU) is performed over USB, different descriptors are used. This document does not cover DFU operation. Therefore, it does not discuss these descriptors and related PS Keys. Refer to separate DFU application notes for more information.

Note:

Some versions of DFU firmware do not support power down in USB suspend: this can cause the suspend current to be above the 2.5mA *USB v2.0 Specification* limit when in DFU mode.

PS Key Name	Location	Default	Description
PSKEY_USB_VERSION	0x02bc	0x0110	Version of the USB specification the device supports (field bcdUSB). Value is stored in Binary Coded Decimal. Older firmware versions default to v1.1 (0x0110). Newer firmware versions default to v2.0 (0x0200). USB compliance tests now require 0x0200.

PS Key Name	Location	Default	Description
PSKEY_USB_DEVICE_CLASS_CODES	0x02bd	0xe0 0x01 0x01	The three bytes contain fields bDeviceClass, bDeviceSubClass, and bDeviceProtocol. Defaults map to WIRELESS_CONTROLLER, RF_CONTROLLER, BLUETOOTH_PROGRAMMING.
PSKEY_USB_VENDOR_ID	0x02be	0x0a12	USB Vendor ID (field idVendor). Defaults to CSR's Vendor ID. This value is used, in combination with the Product ID, to uniquely identify an end product and must be set to the end product manufacturer's ID as per USB certification rules.
PSKEY_USB_PRODUCT_ID	0x02bf	0x0001	USB Product ID (field idProduct). Defaults to CSR's Product ID for Generic Bluetooth devices. This value is used, in combination with the Vendor ID, to uniquely identify an end product and must be set to the end product manufacturer's chosen ID as per USB certification rules.
PSKEY_USB_MANUF_STRING	0x02c1	–	USB Manufacturer text string (index referenced by field iManufacturer). Defaults to Not Present.
PSKEY_USB_PRODUCT_STRING	0x02c2	–	USB Product text string (index referenced by field iProduct). Defaults to Not Present.
PSKEY_USB_SERIAL_NUMBER_STRING	0x02c3	–	USB Serial Number text string (index referenced by field iSerialNumber). Defaults to Not Present.
PSKEY_USB_CONFIG_STRING	0x02c4	-	USB Config text string (index referenced by field iConfiguration). Defaults to Not Present.
PSKEY_USB_ATTRIBUTES	0x02c5	0x00c0	USB Attributes bitmap (field bmAttributes). Bits map to: Bit 7: Reserved (set to one) Bit 6: Self-powered Bit 5: Remote Wake Capable Bits [4:0]: Reserved (set to zero) Defaults to 0xc0: Self-powered, but not Remote Wake Capable.
PSKEY_USB_ATTRIBUTES_POWER	0x03f2	0x0001	A presentation key for bit 7 of USB Attributes bitmap (field bmAttributes). Bit 7 maps to self-powered. Changes to this key are reflected in bit 7 of PSKEY_USB_ATTRIBUTES (location 0x025c) and visa versa. Defaults to TRUE: Self-powered.

PS Key Name	Location	Default	Description
PSKEY_USB_ATTRIBUTES_WAKE	0x03f3	0x0000	A presentation key for bit 6 of USB Attributes bitmap (field bmAttributes). Bit 6 maps to Remote Wake Capable. Changes to this key are reflected in bit 6 of PSKEY_USB_ATTRIBUTES (location 0x025c) and visa versa. Defaults to FALSE: not Remote Wake Capable.
PSKEY_USB_BT_IF_CLASS_CODES	0x02c7	0xe0 0x01 0x01	The three bytes contain fields bInterfaceClass, bInterfaceSubClass, and bInterfaceProtocol for interface 0. Defaults map to WIRELESS_CONTROLLER, RF_CONTROLLER, BLUETOOTH_PROGRAMMING.
PSKEY_USB_LANGID	0x02c9	0x0409	Language ID used in wLANGID field of string descriptors. See USB specification v2.0, section 9.6.7, page 273. Defaults to: Primary: ENGLISH (1) Secondary: ENGLISH_US (9).
PSKEY_USB_BT_SCO_IF_CLASS_CODES	0x02d4	0xe0 0x01 0x01	The three bytes contain fields bInterfaceClass, bInterfaceSubClass, and bInterfaceProtocol for interface 1. Defaults map to WIRELESS_CONTROLLER, RF_CONTROLLER, BLUETOOTH_PROGRAMMING.
PSKEY_USB_ENDPOINT_0_MAX_PACKET_SIZE	0x02d8	0x0040	Maximum packet size for USB endpoint 0 as reported in the bMaxPacketSize0. Only values 8 (0x0008) 16 (0x0010), 32 (0x0020) and 64 (0x0040) are valid.

Table 2.5: PS Keys for USB Descriptors

2.4 Internal Modules, Certification and Non-Spec Compliant Operation

USB device certification tests check a device's compliance with the *USB v2.0 Specification*. The tests are standardised and mandate the use of USB approved connectors. When a device has achieved certification, then the manufacturer has permission to use USB branding, logos, and other intellectual property with that device. The testing ensures that any USB device can be connected to any USB Hub without encountering compatibility problems. Compliance certification must be from an independent body. Obtain it from a USB Plugfest, or from an independent USB test house (see http://www.usb.org for further details).

It is, however, perfectly possible to produce a device that employs a USB interface, but does not receive certification, provided none of the USB organisation's intellectual property, such as logos, are employed when marketing the device. In fact, since testing requires the use of a USB standard connector, if a device does not have a standard connector (e.g. an internal laptop module with a proprietary connector), then it is impossible to obtain USB certification.

This freedom from the need to obtain USB certification for a device if it uses a non-standard connector or that, more broadly, if it will never be plugged into a standard, external, USB port, should not be taken as a license to abuse the USB specification. Following the specification assures a robust and reliable transport protocol: toying with it often results in interoperability problems. Sometimes however, mixing and matching parts of the specification can be useful. Provided the implications of stepping slightly outside the bounds of the specification are well understood, and the entire system is designed to support these slightly non-standard configurations, no problems should be encountered.

This section describes some behaviours defined in the *USB v2.0 Specification* that system designers may want to tweak.

2.4.1 USB VBUS Monitoring

The *USB v2.0 Specification* states that self-powered devices are required to monitor USB VBUS, but that bus-powered devices are not (see section 7.1.5.1, page 141). This is necessary because the specification does not mandate that Hubs must be resistant to latch-up if a voltage is applied to their port pins while powered down. Specifically, if self-powered devices do not check the status of USB VBUS before applying voltage to the USB D+ or USB D- line (in an attempt to initiate enumeration) then the voltage on the signalling line might be enough to latch-up an un-powered hub and subsequently prevent it from powering up correctly.

The same risk does not exist for bus-powered devices: the pull-up voltage is derived from the Hub supplied voltage, so if the Hub is inactive then there is, by definition, no pull-up voltage and no chance of latch-up.

To save on connection pins, or the potential divider components often associated with VBUS monitoring, it may be preferrable to produce a self-powered device that does not monitor USB VBUS. This is permissible provided one of the following two conditions is true:

- The upstream Hub is immune to latch-up.
  Many modern Hubs are designed to be immune to latch-up to protect against poorly configured devices. If the system designer can guarantee that a self-powered module will never be connected to a Hub that is vulnerable to latch-up then there is no need to monitor USB VBUS.
- The module is never powered while the Hub is un-powered.
  Since the point of monitoring USB VBUS is to prevent a powered device from latching-up an un-powered hub, there is no need for such monitoring if the Hub is always powered when the device is powered. If the Hub and the device are on separate power supplies, care must be taken with power supply timing and enable / disable sequences to make sure that the device is always enabled at the same time as or after the Hub, and not merely as part of the same operation.

On BlueCore variants that feature battery charger hardware the charger input is frequently connected to VBUS to allow charging from the USB port. On post BlueCore4 variants which feature battery charger hardware, recent firmware builds allow PSKEY_USB_PIO_VBUS to be set to use the charger input pin to monitor VBUS. This removes the need for a separate PIO to monitor VBUS.

PS Key Name	Location	Default	Description
PSKEY_USB_PIO_VBUS	0x02d1	–	Values between 0 and 15 (or USB_VBUS_VDD_CHG on BlueCore variants with a charger) indicate the PIO line to use to monitor USB VBUS. If key is not present then firmware assumes the USB VBUS is always present.

Table 2.6: PS Key for USB VBUS Monitoring

2.4.2 Suspend Mode Current Draw

The *USB v2.0 Specification* states that bus-powered devices must not draw more than 2.5mA of current from USB VBUS while in suspend mode (see section 7.2.3, page 176 & ECN). This is to protect the upstream Hub from excessive current draw in what is intended to be a low-power state, but it can seriously restrict the functions that a bus-powered device can carry out while Suspended. BlueCore, for example, cannot power its RF synthesiser while in Suspend Mode (see section 4 of this document).

However, if the system designer can guarantee that a module will never be connected to a power supply that is unable to meet its current draw requirements for full operation during Suspend, then it is permissible for the device to maintain full-operation and draw more than the normally permitted 2.5mA while in this mode.

In BlueCore's case one of the main aspects of self-powered operation, the monitoring of USB VBUS, has already been decoupled from the self-powered / bus-powered configuration of the device with the use of PSKEY_USB_PIO_VBUS. Therefore, the behaviour during Suspend is controlled by

PSKEY_USB_ATTRIBUTES_POWER. In effect, if you want to enable a bus-powered device to continue with full, high-current, functionality during Suspend, it should just be configured as a self-powered device. This approach also means that the Host side system is aware of the device's capabilities because the bus-powered / self-powered status of the device is reported during enumeration.

PS Key Name	Location	Default	Description
PSKEY_USB_ATTRIBUTES_POWER	0x03f2	0x0001	A presentation key for bit 7 of USB Attributes bitmap (field bmAttributes). Bit 7 maps to self-powered. Changes to this key are reflected in bit 7 of PSKEY_USB_ATTRIBUTES (location 0x025c) and visa versa. Defaults to TRUE: self-powered.

Table 2.7: PS Key for Self-powered / Bus-powered Configuration

2.4.3 PIO Status in Suspend Mode

To ensure that the limit on current draw in suspend mode for a bus-powered device is met, BlueCore usually sets all PIO lines to low. However, this may not always be the correct for a particular application, so three PS Keys allow the configuration to be set.

PSKEY_USB_SUSPEND_PIO_MASK indicates which PIOs should be set when in suspend mode. A `1` in the mask indicates a PIO line to be set according to the corresponding bits in PSKEY_USB_SUSPEND_PIO_LEVEL and PSKEY_USB_SUSPEND_PIO_DIR; a `0` indicates a PIO line which will be left alone.

For each bit that is set to 1 in PSKEY_SUB_SUSPEND_PIO_MASK, a `0` for the corresponding bit in PSKEY_USB_SUSPEND_PIO_LEVEL indicates that the line should be set low and a `1` that it should be set high. A `0` for the corresponding bit in PSKEY_USB_SUSPEND_PIO_DIR indicates that the line will be set for input, a 1 that it will be set for output.

Note:

If a line is set for input, the level is still useful: it determines whether a weak pull-up or pull-down will be applied.

Any PIO line configured via PSKEY_USB_PIO_PULLUP is handled separately; the bit does not need to be set in any of these three PS Keys.

The keys apply only to a bus-powered USB device; on a self-powered USB device the PIO lines are not modified in suspend mode.

PS Key Name	Location	Default	Description
PSKEY_USB_SUSPEND_PIO_MASK	0x02d7	0xffff	Bit mask of PIOs to be forcibly set when entering Suspend mode as a bus-powered USB device. Defaults to all PIO lines set.
PSKEY_USB_SUSPEND_PIO_DIR	0x02d6	0x0000	Bit mask of whether to set PIOs as inputs or outputs when entering Suspend mode as a bus-powered device. The PIO line must be specified in PSKEY_USB_SUSPEND_PIO_MASK for settings in this PS Key to be effective. A zero (0) indicates input, a one (1) indicates output. Defaults to all PIO lines as inputs.
PSKEY_USB_SUSPEND_PIO_DIR_LEVEL	0x02d5	0x0000	Bit mask of whether to set PIOs high or low (if outputs, with pull-up or pull-down if inputs) when entering Suspend mode as a bus-powered device. The PIO line must be specified in PSKEY_USB_SUSPEND_PIO_MASK for settings in this PS Key to be effective. A zero (0) indicates output low / pull-down, a one (1) indicates output high / pull-up. Defaults to all PIO lines as low / pull-down.

Table 2.8: PS Keys for PIO Settings in Suspend Mode

2.4.4 Resume, Detach and Wake PIOs

Section 2.2 describes USB Suspend and Resume. The signalling for both operations passes over the normal USB D+ and D- lines. In some cases, it can be useful to send these signals, or something similar to them, out-of-band: over additional PIO lines. BlueCore supports this sort of systems setup with three additional out-of-band signals that can each be assigned to their own PIO line: Resume, Detach and Wake.

The first of these signals, Resume, is used to signal that the USB host wakeup from suspend. The PIO line is high to indicate that the host should resume, low otherwise. It remains asserted until activity is restored on the USB. Setting PSKEY_USB_PIO_RESUME is sufficient to enable this feature; notice is taken neither of the remote wakeup setting of PSKEY_USB_ATTRIBUTES nor of whether the host has enabled remote wakeup. If the key is *Not Present* then the feature is not in use.

PIO Resume is often used in place of the in-band bus resume signal for hosts that are unable to respond to the bus signal during suspend because they power down the root hub in suspend in order to save power (e.g. PDAs). The device is typically placed in Suspend using the in-band signal with the PIO Resume signal being routed to an interrupt pin on the host microcontroller; the micro wakes up the USB port resumes the bus when the PIO interrupt pin goes high. While in Suspend the device still maintains a voltage to the USB pull-up resistor, so there is still a current drain of approximately 200μA though it while in this mode, whereas the Detach / Wakeup signalling allows this current draw to be eliminated.

The PIO Detach and PIO Wake signals work together. PIO Detach is similar in function to an out-of-band Suspend signal. When the PIO input goes high BlueCore places the D+ and D- lines in a high impedance state and removes the voltage from the pull-up resistor. This has the same effect as unplugging the device: it drops off the USB bus and the only current draw is that required to run the radio. Radio operation does not cease if already in progress and if activity occurs that generates chip-to-host USB traffic (e.g. an incoming connection request or traffic on an existing link), then the PIO Wake signal is triggered. USB communication can only resume when the PIO Detach signal has been removed, the timing of which is dependant on the host. Figure 2.5 shows BlueCore's timing.

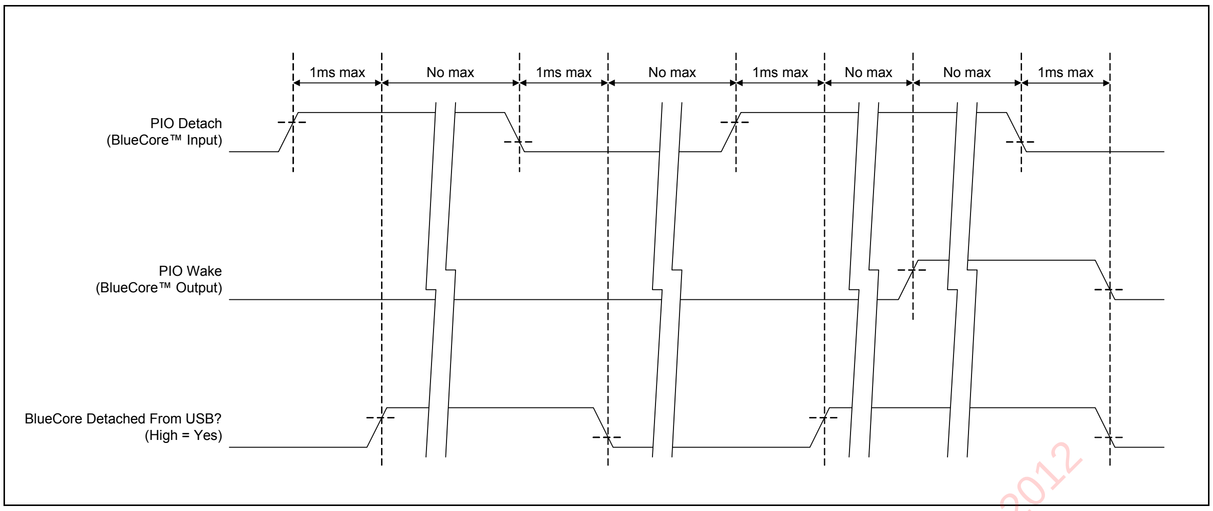


Figure 2.5: PIO Detach and PIO Wake Timing

The PIO Wake signal's duration after each activity that generates chip-to-host traffic can be adjusted using PSKEY_USB_PIO_WAKE_TIMEOUT, which specifies the duration in milliseconds. If the key is not present then PIO Wake is held high indefinitely. This key is of use for hosts that are sometimes unable to respond to the wake signal (e.g. laptops when their lids are closed). If wake is asserted when the host cannot process wake and kept asserted until it is able to process the signal, then the host might be woken up to receive an event which is out of date. The host will, of course, have to process any old events when it does reconnect to a device following a wake timeout.

PIO Resume and PIO Wake can both be in use at the same time. PIO Resume is active in both Suspend and Detach modes. PIO Wake is only active in Detach mode.

PS Key Name	**Location**	**Default**	**Description**
PSKEY_USB_PIO_RESUME	0x02d3	–	PIO line to use for out-of-band Resume signalling. If "Not Present" then this feature is not in use. Defaults to "Not Present".
PSKEY_USB_PIO_DETACH	0x02ce	–	PIO line to use for out-of-band Detach signalling. If the feature is in use and the designated PIO line is high, then USB D+ and D- lines are set to high impedance, voltage is removed from the pull-up resistor and BlueCore effectively drops off the USB bus. If *Not Present* then this feature is not in use. Defaults to *Not Present*.
PSKEY_USB_PIO_WAKEUP	0x02cf	–	PIO used for out-of-band Wake signalling. If the feature is in use and BlueCore is in Detach mode, then each new item of pending chip-to-host traffic causes this line to toggle high for a duration set by PSKEY_USB_PIO_WAKE_TIMEOUT
PSKEY_USB_PIO_WAKE_TIMEOUT	0x02d2	0x0000	The number of seconds for which the PIO Wake signal will be asserted following the generation of data that is to be transmitted to the host. The timeout is reset each time new data is generated. If this value if 0, the signal is asserted indefinitely (or until the host de-asserts detach).

Table 2.9: PS Keys for PIO Resume, Detach and Wake Signalling

3 Electrical Design Guidelines

Although BlueCore is capable of meeting the specification and test requirements of the *USB v2.0 Specification*, CSR cannot guarantee that an application circuit designed around the IC will be compliant. This is because the choice of application circuit, surrounding components and PCB layout all affect USB signal quality and electrical characteristics. The information in this section is a guide that highlights some of the more common problems and how to avoid them. Read this alongside the *USB v2.0 Specification*, with particular attention given to chapter 7. As stated in section 2, independent USB compliance certification must be obtained before an application is deemed USB compliant and can bear the USB logo. Obtain this certification from a USB Plugfest, or from an independent USB test house (see http://www.usb.org for further details).

3.1 Power Supply

The minimum output high voltage level for USB data lines is 2.8V. When supplying 4mA from a data line, the output voltage can fall to VDD-0.2 V. Therefore, to meet the USB specification the voltage on the pad supplying the USB interface (typically VDD_USB, but consult specific device documentation) must be a minimum of 3.0V.

3.2 D+ and D-

The USB data lines emerge as D+ and D-. These pins are connected to the internal USB I/O buffers of BlueCore, and have a low output impedance. To match the connection to the characteristic impedance of the USB cable, series resistors must be included on both D+ and D-. If long (e.g. over 5cm) PCB tracks are use for D+ and D-, the resistors should be within a few centimetres of the BGA package to minimise reflections. The resistors should be of 1% tolerance to provide good symmetry of D+ and D- signal waveforms. This minimises common-mode noise emissions during differential signalling.

Since the input impedance seen by the cable is affected by IC characteristics, track layout and connector the discrete resistor value required may vary between 27 and 39 with 33 being nominal (cable impedance is approximately 40 , see USB cable specification for details). If the resistance is too low signal overshoot occurs; if it is too high, the slew rate falls below specification causing undershoot.

The D+ and D- pins adhere to the *USB v2.0 Specification*, chapter 7, Electrical Requirements. (For specifications of other PIO based pins, see the PIO specification in a BlueCore datasheet.)

3.3 PCB Tracks

The PCB tracks for D+ and D- should have a nominal impedance of 45 ohms +/-15%. Ensure that both D+ and D- tracks are of the same length and lie alongside one another. CSR recommends an impedance differential from track-to-track of 90 ohms +/-15%. CSR strongly recommends the use of a ground plane.

3.4 Ferrite Beads

Avoid the use of ferrite-beads or other inductors on D+ and D-. The most common mistake made by engineers that causes their designs to fail USB compliance testing is passing D+ and D- through a ferrite bead. The use of ferrite beads limits common mode noise. If USB were a purely differential signalling scheme then there would be no negative effect. However, USB signalling is not always differential: single-ended zeros (SE0s) pull both D+ and D- low to delimit packets. The common-mode element of the SE0 is distorted by the inductance of the ferrite bead. Passing USB GND through an inductor causes the same problem since it provides the return current path for D+ and D-.

Figure 3.1 shows the common-mode slew rate limitation imposed by the interposition of a single ferrite on D+ and D-, and the subsequent violation of the USB signal quality. The slew rate and amplitude limits are marked in red. To meet the *USB v2.0 Specification* all signals should be within the limits; traces in the red area indicate failure. Although the differential signals pass specification, the ferrite causes the highlighted single-ended-zero to fail. (The diagram was generated by the MatLab test scripts provided by the USB implementers' forum. The test scripts are available from http://www.usb.org.)

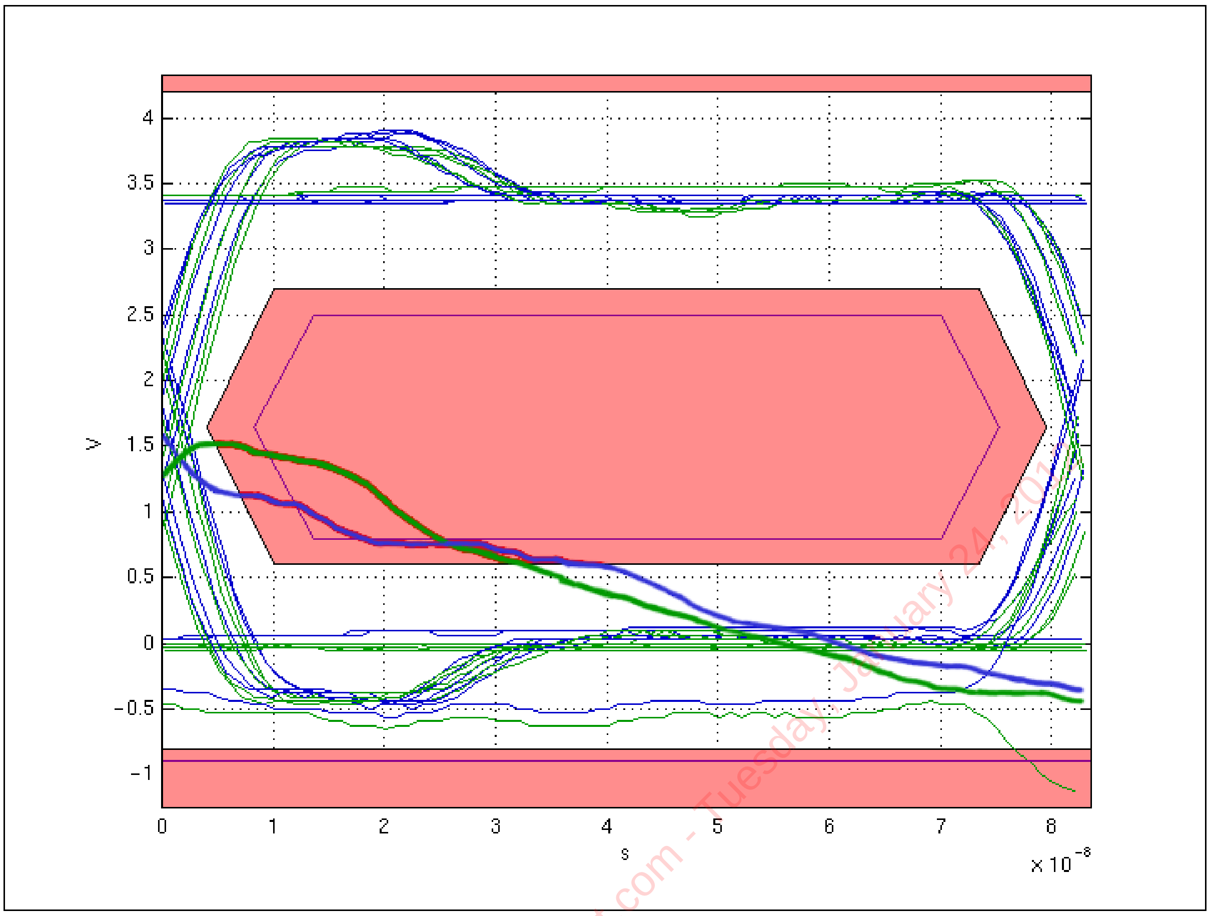


Figure 3.1: Effect of Ferrites on Single Ended Zero Signalling

If EMI must be reduced to meet FCC or EMI requirements, then edge-rate control capacitors may be added as a last resort between D+ and D- and ground. Typically, a few pF should be applied to limit the slew rate. Both capacitors should be of the same value. A preferable technique is to improve screening, for example by passing D+ and D- traces between ground planes or power planes on the PCB.

4 USB Suspend and Bluetooth Low Power Modes

Section 2.2 outlined the relevant basics of USB Suspend operation. This section describes in detail how USB Suspend affects Bluetooth operation.

The reason for placing a device into USB Suspend is to save power. Full speed USB (the speed used by Bluetooth devices) is a relatively fast bus, running at 12MHz, but asynchronously, so a 48MHz clock is required to receive it reliably. Fast clocks draw a lot of power, and keeping a 48MHz clock running at all times may not be desirable for battery powered devices. On BlueCore2 for example, maintaining the required circuitry at a sufficiently high speed requires around 10mA of current, even if the Bluetooth radio itself is doing nothing. Putting the device into Suspend allows the use of the Deep Sleep mode.

The Deep Sleep current draw will be the only current draw if no radio activity is required. However, provided that a device is configured as self-powered (see below for details), then radio activity is permitted while in Suspend mode. If a device is in a low duty-cycle Page Scan mode, for example, it will carry out the scan then return to Deep Sleep in-between: on BlueCore2 this mode draws about 0.6mA.

There are four modes of operation for a system that implements USB suspend, each of which can be added on top of the previous one and each of which brings an additional level of complexity:

1. Global Suspend. Suspend the device when both the Bluetooth device and any others on the USB bus are completely idle (i.e., no radio activity). This mode is often used when the entire system enters a low-power mode.
2. Selective Suspend. Suspend the Bluetooth device when it is completely idle (i.e., no radio activity), but permit other devices on the USB bus to remain active if necessary. Only the host can bring the device out of Suspend.
3. Selective Suspend with Remote Wake. Suspend the Bluetooth device during periods of low USB activity. In this mode of operation, the device can initiate a Remote Wake when it generates chip-to-host USB traffic, but the host is never in a low-power mode itself when this happens.
4. Wake On Bluetooth. This is the most complicated mode. It permits Bluetooth activity and a Remote Wake procedure to wake a system from a low-power mode.

4.1 Global Suspend

Global suspend requires no special configuration on the Bluetooth device. The Global Suspend state is typically initiated by the System entering a low power state (e.g. PC Suspend). Prior to the Global Suspend of the USB bus the Host Bluetooth stack should close all open connections and cancel any Paging, Inquiry or Scans so that there is no chance that the Bluetooth device will generate USB traffic while in Suspend. (If the device is configured to be bus-powered, placing the device in Suspend mode will automatically halt all radio activity, effectively removing the chance of any chip-to-host traffic being generated, but it is still better to close any links gracefully rather than letting them timeout.)

Entry summary:

1. System indicates intent to enter low power state.
2. Host Bluetooth stack closes connections; cancels Paging, Inquiry, Scanning.
3. Entire USB bus is Suspended.
4. System enters low power state.

Exit summary:

1. System exits low power state.
2. Entire USB bus Resumes.
3. Normal operation.

4.2 Selective Suspend

Selective Suspend requires no special configuration of BlueCore. Then mode is typically entered on an opportunistic basis: if the Bluetooth device is not being used, then it is better for it to draw less power.

Entry summary:

1. Device is idle: no open connection, no Paging, Inquiry, Scanning.
2. Host Bluetooth stack places Bluetooth device in Selective Suspend.

Exit summary:

1. Host Bluetooth stack needs to use Bluetooth device.
2. Host Bluetooth stack issues Resume to Bluetooth device.
3. Normal operation.

Although the summary contains fewer steps than that for Global Suspend, it is often more involved since placing the Bluetooth device into Selective Suspend and issuing the Resume instruction requires system level support, which not all systems can provide (e.g. versions of Microsoft Windows prior to XP).

4.3 Selective Suspend with Remote Wake

To support Remote Wake, configure BlueCore appropriately: set both PSKEY_USB_ATTRIBUTES_WAKE (location 0x03f3) and PSKEY_USB_ATTRIBUTES_POWER (location 0x03f2) to TRUE (0x0001). The former reports to the host that the device supports Remote Wake; if the device does not report this feature then the host makes no attempt to enable it. The latter configures the device as self-powered.

Section 2.2 of this document describes the 2.5mA current draw restriction placed on USB bus-powered devices in suspend mode. This is insufficient to enable BlueCore to power its radio section. Without radio activity there can be no over-air communication that might trigger chip-to-host traffic. While it is possible to enable Remote Wake operation for a bus-powered device, for a Bluetooth radio such a configuration effectively makes Remote Wake redundant. Therefore, self-powered is the only useful Remote Wake enabled configuration. See section 2.2 and section2.4 of this document for details on specification compliant and non-specification compliant self-powered configurations.

Take care when designing a system that will use Remote Wake to ensure the entire system will support the signalling, including the Root Hub and any USB Hubs that may be between the Bluetooth device and the Root Hub (some Hubs do not pass on the Remote Wake signalling correctly).

Like Selective Suspend on its own, Selective Suspend with Remote Wake is normally entered on an opportunistic basis: if the Bluetooth device is in a low activity state, then it is placed in Selective Suspend. Radio activity continues uninterrupted.

Selective Suspend can be exited in one of two ways. The host may generate host-to-chip traffic, in which case the Resume operation is the same as for Selective Suspend (see section 3). However, with the device configured for self-powered operation with Remote Wake, radio activity can also generate chip-to-host traffic, in which case the chip issues a Remote Wake signal. The Remote Wake signal propagates back up to the Root Hub and the Host, which then issues a Resume to the device. The device can then send its chip-to-host traffic.

It takes longer to carry out a Remote Wake or Resume operation than it does just to start communications over an active USB bus, so there will be an increase in latency for devices that take advantage of this mode. However, the latency increase is in the order of milliseconds and the power savings are usually worth the price. There is a design decision to be make on how aggressive to be about placing the Bluetooth device into Selective Suspend mode: the more aggressive the greater the power savings, but the higher the latency penalty. Depending on the system design priorities, placing the Bluetooth device into Selective Suspend after one to ten seconds of no USB communication is normally a sensible choice.

If a Host wishes to be very aggressive about saving power then it may be beneficial to remove power from the USB Root Hub port. See section 2.4.4 for details of the PIO Resume, PIO Detach and PIO Wake signals that can allow Selective Suspend with Remote Wake mode to work on such systems.

Entry summary:

1. No USB communication with device for X seconds.
2. Host Bluetooth stack enables Remote Wake feature on Bluetooth device.
3. Host Bluetooth stack places Bluetooth device in Selective Suspend.

Exit summary (for chip-to-host traffic):

1. Radio activity generates chip-to-host traffic.
2. Bluetooth device issues Remote Wake signal to upstream Hub.
3. Remote Wake signal is propagated to Root Hub and Host Bluetooth stack.
4. Host Bluetooth stack issues Resume signal to Bluetooth device.
5. Bluetooth device Resumes and sends chip-to-host traffic.
6. Normal operation.

4.4 Wake on Bluetooth

No additional configuration is required for BlueCore to implement Wake on Bluetooth beyond those for Selective Suspend with Remote Wake. The only difference between the two modes is the state the Host is in: in Selective Suspend with Remote Wake the host is active, in the Wake on Bluetooth mode the Host is in a low-power mode (from with it can be woken).

Wake on Bluetooth adds four complications to Selective Suspend with Remote Wake. All of these are on the system side, not on BlueCore:

1. Hardware Design: if the Remote Wake signal is to be passed from Bluetooth device to Root Hub while in a system-wide low-power mode, any intermediate Hubs must remain powered while in the low-power mode. Hubs are often powered down in low-power modes precisely to save power. It is therefore often better to connect the Bluetooth device directly to the Root Hub. PIO Resume, PIO Detach and PIO Wake signals can also be used (see section 2.4.4 of this document).
2. What should be done with existing connections on entry? Should they be maintained, or dropped? If they are maintained, what happens if the remote side disconnects?
3. Which remote Bluetooth devices can wake up the system? All devices? Or only some? Should they be particular known devices? Or entire classes of devices? And is this list of “permitted” devices fixed, or will it change?
4. How can unnecessary system wake-up events be prevented? Assuming that the system should not wake up on all possible events, how should the system be configured to reduce the number of unnecessary

Points 2 - 4 can be addressed by developing a policy on which devices can wake the system from its low-power state; how to set up the device for Wake on Bluetooth operation and filter communications from the device so that only the essential ones get through; and finally, how to behave when the system is woken up.

Entry in to the Wake on Bluetooth mode is typically initiated by the system entering a low-power mode (e.g. PC Suspend).

4.4.1 Permitted Devices

Before entering Wake on Bluetooth mode it is necessary to decide which devices are permitted to initiate a Wake event. When this list is established it is used to set up the Bluetooth device correctly prior to placing it in suspend.

One of the main decisions to make is whether the device should be Discoverable so that any remote device can connect to it and initiate a Wake, or merely Connectable so that only remote devices that already know about the device can wake it. Although it is a Page, not an Inquiry that initiates a Wake and it is possible to place a device in a mode where it can be Discovered via an Inquiry, but can not be connected to via a Page, this is neither useful nor sensible. When searching for Bluetooth devices it is standard practise to follow up an Inquiry responses with a Remote Name Request, which relies on the Page procedure. It is also common to follow this up with an SDP connection to find out what services are offered. A device that only responds to Inquiries will thus present confusing information to a remote device that discovers it and only frustrates a user that tries to gather more information.

Placing a device in Page / Inquiry scan, so that it is Connectable and Discoverable draws twice as much power as placing it in Page scan (Connectable) only. Although the current draw involved is only a few hundred milliamps, this may influence the decision for some applications.

If the decision is taken to limit the device to Page Scan only in the low-power state then a further choice can be taken: permit any remote device to connect; permit only certain classes of device to connect; permit only certain devices to connect. (The resolution of these choices is determined by the HCI Set Event Filter and HCI Set Event Mask commands, so it is not possible to, for example, specify a range of Bluetooth device addresses that can connect; each Bluetooth device address must be specified individually. See Part E, sections 7.3.1 and section 7.3.3, pages 440 and 443 respectively, of the *Bluetooth Core Specification* v1.2 for details.)

Taking the PC as an example, the decision might be taken to only permit mice and keyboards to wake the PC, or the specific mouse and keyboard that are connected when the low-power state is entered.

The list of permitted devices may change depending on the system's configuration. For example, on a laptop PC it might be ill advised to permit a mouse to wake a laptop up when the lid is closed since the machine could be liable to overheating in the confines of a briefcase; the mouse might only be permitted to wake the laptop when the lid is open.

4.4.2 Setup Prior to Selective Suspend

Prior to entering Selective Suspend, configure Event Filters and Event Masks and disconnect any remote devices that are not on the permitted list.

The HCI Set Event Mask and HCI Set Event Filter commands are described in Part E, sections 7.3.1 and section 7.3.3, pages 440 and 443 respectively, of the *Bluetooth Core Specificatio*n v1.2.

If the list of permitted devices is selective according to Class of Device or Bluetooth device address, set the Event Filter to automatically accept connections from any of the permitted devices and the Event Mask should be set to suppress the Connection Request Event. This combination means that connections from permitted devices are automatically accepted (generating a Connection Complete Event that will wake the system) while incoming connections from other devices will be ignored.

A decision must be taken on whether to disconnect permitted devices prior to entering Selective Suspend and let a reconnection from them wake the system, or to leave the connection intact. If the connection is left intact, then the suppression of other Events using the HCI Set Event Mask command should be considered to minimise the number of unnecessary Wakes.

For example, returning again to the PC with a mouse connected: Bluetooth mice typically negotiate longer Sniff intervals as they are left idle before eventually disconnecting. To avoid waking the PC every time the Sniff interval is changed it is necessary to suppress the HCI Mode Change event. To avoid waking the PC when the mouse disconnects, the HCI Disconnection Complete Event must be suppressed.

If an event for an active connection is suppressed prior to entering Selective Suspend, the Host Bluetooth stack must check on the status of the link after USB communications have been resumed. In the above example sending a restrictive HCI Write Link Policy command to the appropriate ACL connection handle would have the effect of placing the link in a known state and checking that the connection was still in place: if it had been silently dropped then the command would return the appropriate error code.

When the Bluetooth device is correctly configured, Selective Suspend mode can be entered in the same way as before.

4.4.3 Summary

Entry summary:

1. System indicates intent to enter low power state.
2. Host Bluetooth stack configures Bluetooth device according to permitted device list.
3. Entire USB bus is Suspended.
4. System enters low power state.

Exit summary (for chip-to-host traffic):

1. Radio activity generates chip-to-host traffic. (Some radio activity is filtered out, so only permitted devices can wake the system.)
2. Bluetooth device issues Remote Wake signal to upstream Hub.
3. Remote Wake signal is propagated to Root Hub and Host Bluetooth stack.
4. Host Bluetooth stack issues Resume signal to Bluetooth device.
5. Bluetooth device Resumes and sends chip-to-host traffic.
6. Normal operation.

5 Battery Charging from USB

The USB VBUS supply is often used to charge on-board batteries. This was previously often done in a non-compliant manner, but the USB-IF has now issued a specification for charging batteries, which clarifies behaviour required from USB chargers and battery powered devices. Listed below are the main provisions in this specification:

- Dedicated charger: A device capable of supplying USB VBUS but not capable of enumerating downstream devices. Indicated by charger shorting its D+ & D- lines together.
- Dead Battery provision: This describes how a peripheral should act when its battery is insufficiently charged to allow proper enumeration.

5.1.1 Dead Battery Provision

Section 2 of the *Battery Charging Specification*, Revision 1.0, states that while peripherals may normally only draw 2.5mA before connection and in suspend, portable devices may draw 100mA until they are able to correctly enumerate, effectively removing the time specification between USB Attach and USB Connect. This behaviour is only allowed until the device's battery has charged sufficiently to allow the device to enumerate. It is specifically excluded to use this time for other purposes such as charging the battery above the weak battery threshold, making phone calls, playing media or establishing a wireless connection. Use of the Dead Battery Provision must be specified for compliance testing. Consult the *Battery Charging Specification* for a full description of this mode.

5.1.2 Charge Currents

Most BlueCore devices allow the charge current to be varied. Use this facility to keep the charge current within specification: before enumeration and after enumeration as a low-power device the total USB VBUS current must be below 100mA. After enumeration, the current may be increased up to the current specified in the USB descriptor (max 500mA).

5.1.3 Charging in Suspend

CSR Unified Firmware v23c and later automatically disables the charger when entering USB suspend, and re-enables it on USB resume when configured as a bus powered device. When configured as a self-powered device the VM application receives messages informing it of USB suspend and resume, and the VM application can decide on any action required.

5.1.4 USB VBUS Voltage Consideration

In worst case conditions the USB VBUS supply may fall as low as 4.4V. This is likely to give insufficient headroom for the charger circuit to give a full charge to 4.2V. In this case charging may terminate early.

6 USB Termination When Interface not in Use

To obtain minimum current consumption on devices prior to BlueCore5 when VDD_USB is powered but the USB block is unused (typically when UART is used) the USB DP and DN pins should be weakly pulled low externally. This ensures the USB pins do not float and waste current through spurious transitions. BlueCore5 and later devices do not require external termination.

Appendix A USB Persistent Store Keys

PS Key Name	Location	Default	Description
PSKEY_HOST_INTERFACE	0x01f9	0x0001	Change from default of 1 (BCSP) to 2 (USB) to enable USB interface.
PSKEY_HOST_INTERFACE_PIO_USB	0x0250	–	Set to a value between 0 and 15 to force the use of the USB interface when that PIO line pulled high, overriding PSKEY_HOST_INTERFACE.
PSKEY_USB_MAX_POWER	0x02c6	0x0000	Maximum current draw of device in units of 2mA. Keep at default of 0 for low-power and self-powered devices (where max current draw is <100mA).
PSKEY_USB_PIO_VBUS	0x02d1	–	Values between 0 and 15 indicate PIO line to use to monitor USB VBUS. If key is not present then firmware assumes that USB VBUS is always present. Keep at default (not present) for bus-powered devices.
PSKEY_USB_PIO_PULLUP	0x02d0	16	Values between 0 and 15 indicate PIO line to use to enable and disable USB D+/D- pull-up resistor. If key is not present then the firmware will not use any PIO line. The value of decimal 16 is a "magic number" that enables the use of an internal pull-up on the USB D+ line. Keep at default of 16 for bus powered devices.
PSKEY_USB_PIO_RESUME	0x02d3	–	PIO line to use for out-of-band Resume signalling. If "Not Present" then this feature is not in use. Defaults to "Not Present".
PSKEY_USB_PIO_DETACH	0x02ce	–	PIO line to use for out-of-band Detach signalling. If the feature is in use and the designated PIO line is high, then USB D+ and D- lines are set to high impedance, voltage is removed from the pull-up resistor and BlueCore effectively drops off the USB bus. If "Not Present" then this feature is not in use. Defaults to "Not Present".
PSKEY_USB_PIO_WAKEUP	0x02cf	–	PIO used for out-of-band Wake signalling. If the feature is in use and BlueCore is in Detach mode, then each new item of pending chip-to-host traffic cause this line to toggle high for a duration set by PSKEY_USB_PIO_WAKE_TIMEO UT

PS Key Name	Location	Default	Description
PSKEY_USB_PIO_WAKE_TIMEOUT	0x02d2	0x0000	The number of seconds for which the PIO Wake signal will be asserted following the generation of data that is to be transmitted to the host. The timeout is reset each time new data is generated. If this value if 0, the signal will be asserted indefinitely (or until the host de-asserts detach).
PSKEY_USB_SUSPEND_PIO_MASK	0x02d7	0xffff	Bit mask of PIOs to be forcibly set when entering Suspend mode as a bus-powered USB device. Defaults to all PIO lines set.
PSKEY_USB_SUSPEND_PIO_DIR	0x02d6	0x0000	Bit mask of whether to set PIOs as inputs or outputs when entering Suspend mode as a bus-powered device. The PIO line must be specified in PSKEY_USB_SUSPEND_PIO_MASK for settings in this PS Key to be effective. A zero (0) indicates input, a one (1) indicates output. Defaults to all PIO lines as inputs.
PSKEY_USB_SUSPEND_PIO_DIR_LEVEL	0x02d5	0x0000	Bit mask of whether to set PIOs high or low (if outputs, with pull-up or pull-down if inputs) when entering Suspend mode as a bus-powered device. The PIO line must be specified in PSKEY_USB_SUSPEND_PIO_MASK for settings in this PS Key to be effective. A zero (0) indicates output low / pull-down, a one (1) indicates output high / pull-up. Defaults to all PIO lines as low / pull-down.
PSKEY_USB_VERSION	0x02bc	0x0200	Version of the USB specification the device supports (field bcdUSB). Value is stored in Binary Coded Decimal. Older firmware versions default to v1.1 (0x0110). Newer firmware versions default to v2.0 (0x0200). USB compliance tests now require 0x0200.
PSKEY_USB_DEVICE_CLASS_CODES	0x02bd	0xe0 0x01 0x01	The three bytes contain fields bDeviceClass, bDeviceSubClass, and bDeviceProtocol. Defaults map to WIRELESS_CONTROLLER, RF_CONTROLLER, BLUETOOTH_PROGRAMMING.
PSKEY_USB_VENDOR_ID	0x02be	0x0a12	USB Vendor ID (field idVendor). Defaults to CSR's Vendor ID. This value is used, in combination with the Product ID, to uniquely identify an end product and must be set to the end product manufacturer's ID as per USB certification rules.

PS Key Name	Location	Default	Description
PSKEY_USB_PRODUCT_ID	0x02bf	0x0001	USB Product ID (field idProduct). Defaults to CSR's Product ID for Generic Bluetooth devices. This value is used, in combination with the Vendor ID, to uniquely identify an end product and must be set to the end product manufacturer's chosen ID as per USB certification rules.
PSKEY_USB_MANUF_STRING	0x02c1	–	USB Manufacturer text string (index referenced by field iManufacturer). Defaults to Not Present.
PSKEY_USB_PRODUCT_STRING	0x02c2	–	USB Product text string (index referenced by field iProduct). Defaults to Not Present.
PSKEY_USB_SERIAL_NUMBER_STRING	0x02c3	–	USB Serial Number text string (index referenced by field iSerialNumber). Defaults to Not Present.
PSKEY_USB_CONFIG_STRING	0x02c4	-	USB Config text string (index referenced by field iConfiguration). Defaults to Not Present.
PSKEY_USB_ATTRIBUTES	0x02c5	0x00c0	USB Attributes bitmap (field bmAttributes). Bits map to: Bit 7: Reserved (set to one) Bit 6: Self-powered Bit 5: Remote Wake Capable Bits 0 – 4: Reserved (set to zero) Defaults to 0xc0: Self-powered, but not Remote Wake Capable.
PSKEY_USB_ATTRIBUTES_POWER	0x03f2	0x0001	A presentation key for bit 7 of USB Attributes bitmap (field bmAttributes). Bit 7 maps to Self-powered. Changes to this key are reflected in bit 7 of PSKEY_USB_ATTRIBUTES (location 0x025c) and visa versa. Defaults to TRUE: Self-powered.
PSKEY_USB_ATTRIBUTES_WAKE	0x03f3	0x0000	A presentation key for bit 6 of USB Attributes bitmap (field bmAttributes). Bit 6 maps to Remote Wake Capable. Changes to this key are reflected in bit 6 of PSKEY_USB_ATTRIBUTES (location 0x025c) and visa versa. Defaults to FALSE: not Remote Wake Capable.
PSKEY_USB_BT_IF_CLASS_CODES	0x02c7	0xe0 0x01 0x01	The three bytes contain fields bInterfaceClass, bInterfaceSubClass, and bInterfaceProtocol for interface 0. Defaults map to WIRELESS_CONTROLLER, RF_CONTROLLER, BLUETOOTH_PROGRAMMING.

PS Key Name	Location	Default	Description
PSKEY_USB_LANGID	0x02c9	0x0409	Language ID used in wLANGID field of string descriptors. See USB specification v2.0, section 9.6.7, page 273. Defaults to: Primary: ENGLISH (1) Secondary: ENGLISH_US (9).
PSKEY_USB_BT_SCO_IF_CLASS_CODES	0x02d4	0xe0 0x01 0x01	The three bytes contain fields bInterfaceClass, bInterfaceSubClass, and bInterfaceProtocol for interface 1. Defaults map to WIRELESS_CONTROLLER, RF_CONTROLLER, BLUETOOTH_PROGRAMMING.
PSKEY_USB_ENDPOINT_0_MAX_PACKET_SIZE	0x02d8	0x0040	Maximum packet size for USB endpoint 0 as reported in the bMaxPacketSize0. Only values 8 (0x0008) 16 (0x0010), 32 (0x0020) and 64 (0x0040) are valid.
PSKEY_USB_BCD_DEVICE	0x007D	buildid	The bcdDevice field of the Device Descriptor
PSKEY_USB_HOST_WAKE_ENABLE	0x01eb	0x0004	Configures controller to wake host with an out of band PIO signal
PSKEY_USB_HOST_WAKE_TIME	0x01ec	500000 500 0	Controls the host wake timings
PSKEY_USB_HOST_WAKE_SIGNAL	0x01ed	0x0031	Controls the host wake PIO and polarity
PSKEY_USB_DATA_PLUS_PULL_CONTROL	0x01f0	AS_SOON_AS_POSSIBLE	Controls when BlueCore will attach the pullup to the D+ line.
PSKEY_USB_CONFIG	0x02d9	0x30	Only change on advice from CSR
PSKEY_USB_STRING_DESCRIPTORS_MAP	0x02da	-	Maps string descriptor index numbers sent in requests from the host, to PSKEY numbers which hold the response to return to the host .
PSKEY_USB_STRING_DESCRIPTOR0	0x02db	-	Can be used to hold the response to a string descriptor request from the host. There are 16 of these PSKEYS in total.
PSKEY_USB_FEATURE_DESC_MAP	0x02eb	-	Maps USB feature requests which are sent by some host stacks, to PSKEY numbers which hold the response to return to the host.
PSKEY_USB_FEATURE_DESC0	0x02ec	-	Can be used to hold the response to a feature descriptor request from the host. There are 16 of these PSKEYS in total.

PS Key Name	Location	Default	Description
PSKEY_USB_ALLOW_DEEP_SLEEP	0x02fc	0x0001	Controls which USB states BlueCore will deep sleep in. Defaults to SUSPEND only, but can be configure to deep sleep when DETACHED from the bus as well.
PSKEY_USB_VM_CONTROL	0x03c0	0x0000	When FALSE BlueCore will be configure to enumerate as an HCI BlueTooth device. When TRUE the VM is in control of the USB interface and BlueCore may be configured to enumerate as another device class.
PSKEY_USB_DONT_RESET_BOOTMODE_ON_HOST_RESET	0x03B9	0x0000	Controls which boot mode BlueCore we use when it receives a USB reset.

Table A.1: USB PS Keys

Terms and Definitions

ACL	Asynchronous ConnectionLess
BGA	Ball Grid Array
BlueCore®	Group term for CSR's range of Bluetooth wireless technology chips
Bluetooth®	Set of technologies providing audio and data transfer over short-range radio connections
CSR	Cambridge Silicon Radio
DFU	Device Firmware Upgrade
EMI	ElectroMagnetic Interference
FCC	Federal Communications Commission
HCI	Host Controller Interface
IC	Integrated Circuit
PCB	Printed Circuit Board
PDA	Personal Digital Assistant
PIO	Programmable Input/Output
PS Key	Persistent Store Key
SCO	Synchronous Connection-Oriented
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VM	Virtual Machine

Document References

Document	**Reference**
Booting BlueCore ROM	CS-101436-ME
Specification of the Bluetooth System	v1.0 and v1.2
Universal Serial Bus Battery Charging Specification	V1.0 (March 8, 2007)
Universal Serial Bus Specification & associated Engineering Change Notices (ECN)	v2.0