WT USB

Design Guide

Version 1.6

Thursday, August 16, 2007

blue giga
TECHNOLOGIES

Copyright © 2000-2007 Bluegiga Technologies

All rights reserved.

Bluegiga Technologies assumes no responsibility for any errors, which may appear in this manual. Furthermore, Bluegiga Technologies reserves the right to alter the hardware, software, and/or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein. Bluegiga Technologies' products are not authorized for use as critical components in life support devices or systems.

The WRAP is a registered trademark of Bluegiga Technologies

The *Bluetooth* trademark is owned by the *Bluetooth* SIG Inc., USA, and is licensed to Bluegiga Technologies.

All other trademarks listed herein are owned by their respective owners.

Contents:

List of Figures:

VERSION HISTORY

Version	Date	Author	Comments
1.0	1.1.2005	*MSa*	*First Version*
1.1	3.1.2005	*MSa*	*Self powered schematics added*
1.2	11.1.2006	*MSa*	*Parameter descriptions added*
1.3	12.1.2006	*MSa*	*Software approaches added*
1.4	13.1.2006	*MSa*	*TR's comment added*
1.5	28.4.2006	*MSa*	*Default parameters added*
1.5	16.8.2007	*MSa*	*Default parameters fixed*

TERMS & ABBREVIATIONS

Term or Abbreviation	Explanation
Bluetooth	Set of technologies providing audio and data transfer over short-range radio connections
CSR	Cambridge Silicon Radio
HCI	Host Controller Interface
iWRAP	Interface for WRAP
PIO	Peripheral Input Output
USB	Universal Serial Bus
VDD	Voltage Input

VM	Virtual Machine
WRAP	Wireless Remote Access Platform

1. INTRODUCTION

This document contains basic information about the USB interface in WT Bluetooth® modules. The document also instructs how to do self powered or bus powered USB design, what parameters are necessary to modify and what software components are needed.

2. USB INTERFACE

WT modules contain a full speed (12Mbits/s) USB interface that is capable of driving a USB cable directly. No external USB transceiver is required. The device operates as a USB peripheral, responding to requests from a master host controller such as a PC. Both the OHCI and the UHCI standards are supported. The set of USB endpoints implemented can behave as specified in the USB section of the Bluetooth v2.0 + EDR specification or alternatively can appear as a set of endpoint appropriate to USB audio devices such as speakers.

As USB is a Master/Slave oriented system (in common with other USB peripherals), WT modules only supports USB Slave operation.

2.1 USB Data Connections

The USB data lines emerge as pins USB_DP and USB_DN. These terminals are connected to the internal USB I/O buffers of the WT Bluetooth modules and therefore have low output impedance. To match the connection to the characteristic impedance of the USB cable, resistors are included with USB_DP / USB_DN and the cable.

2.2 USB Pull-Up Resistor

WT modules feature an internal USB pull-up resistor. This pulls the USB_DP pin weakly high when WT module is ready to enumerate. It signals to the PC that it is a full speed (12Mbit/s) USB device. The USB internal pull-up is implemented as a current source, and is compliant with Section 7.1.5 of the USB specification v1.2. The internal pull-up pulls USB_DP high to at least 2.8V when loaded with a 15k +/-5% pull-down resistor (in the hub/host) when VDD=3.3V. This presents a therein resistance to the host of at least 900Ohms. Alternatively, an external 1.5K pull-up resistor can be placed between a PIO line and D+ on the USB cable. The firmware must be alerted to which mode is used by setting PS Key PSKEY_USB_PIO_PULLUP appropriately. The default setting uses the internal pull-up resistor.

2.3 Power Supply

The USB specification dictates that the minimum output high voltage for USB data lines is 2.8V. To safely meet the USB specification, the voltage on the VDD_USB supply terminals must be an absolute minimum of 3.1V. 3.3V is recommended for optimal USB signal quality.

2.4 Suspend Current

All USB devices must permit the USB controller to place them in a USB Suspend mode. While in USB Suspend, bus powered devices must not draw more than 0.5mA from USB VBUS (self powered devices may draw more than 0.5mA from their own supply). This current draw requirement prevents operation of the radio by bus powered devices during USB Suspend.

The voltage regulator circuit itself should draw only a small quiescent current (typically less than 100uA) to ensure adherence to the suspend current requirement of the USB specification. This is not normally a problem with modern regulators. Ensure that external LEDs and/or amplifiers can be turned off by WT module. The entire circuit must be able to enter the suspend mode. (For more details on USB Suspend, see separate CSR documentation).

2.5 Detach and Wake-Up Signaling

WT modules can provide out-of-band signaling to a host controller by using the control lines called 'USB_DETACH' and 'USB_WAKE_UP'. These are outside the USB specification (no wires exist for them inside the USB cable), but can be useful when embedding a WT module into a circuit where no external USB is visible to the user. Both control lines are shared with PIO pins and can be assigned to any PIO pin by setting the PS Keys PSKEY_USB_PIO_DETACH and PSKEY_USB_PIO_WAKEUP to the selected PIO number.

USB_DETACH is an input which, when asserted high, causes the WT module to put USB_DN and USB_DP in high impedance state and turned off the pull-up resistor on D+. This detaches the device from the bus and is logically equivalent to unplugging the device. When USB_DETACH is taken low, the WT module will connect back to USB and await enumeration by the USB host.

USB_WAKE_UP is an active high output (used only when USB_DETACH is active) to wake up the host and allow USB communication to recommence. It replaces the function of the software USB WAKE_UP message (which runs over the USB cable), and cannot be sent while WT module is effectively disconnected from the bus.

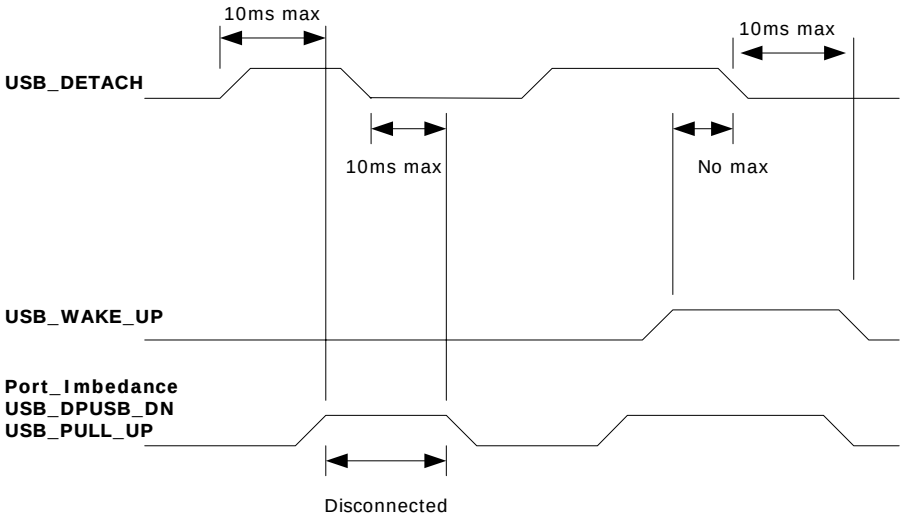

Figure 1: USB_DETACH and USB_WAKE_UP Signal

2.6 USB 1.1 Compliance

WT modules are qualified to the USB specification v1.1, details of which are available from http://www.usb.org. The specification contains valuable information on aspects such as PCB track impedance, supply inrush current and product labeling.

Although WT mdodules meets the USB specification, CSR cannot guarantee that an application circuit designed around the chip is USB compliant. The choice of application circuit, component choice and PCB layout all affect USB signal quality and electrical characteristics. The information in this document is intended as a guide and should be read in association with the USB specification, with particular attention being given to Chapter 7. Independent USB qualification must be sought before an application is deemed USB compliant and can bear the USB logo. Such qualification can be obtained from a USB plug fest or from an independent USB test house.

Terminals USB_DP and USB_DN adhere to the USB specification 2.0 (Chapter 7) electrical requirements.

2.7 USB 2.0 Compatibility

WT modules are compatible with USB v2.0 host controllers; under these circumstances the two ends agree the mutually acceptable rate of 12Mbits/s according to the USB v2.0 specification.

3. BUS POWERED MODE

In bus powered mode the application circuit draws its current from the 5V VBUS supply on the USB cable. WT module negotiates with the PC during the USB enumeration stage about how much current it is allowed to consume.

WT12:

For Bluetooth applications, it is recommended that the regulator used to derive 3.3V from VBUS is rated at 100mA average current and should be able to handle peaks of 120mA without fold back or limiting. In bus powered mode, WT12 module requests 100mA during enumeration.

WT11:

For Bluetooth applications, it is recommended that the regulator used to derive 3.3V from VBUS is rated at 200mA average current and should be able to handle peaks of 220mA without fold back or limiting.

When selecting a regulator, be aware that VBUS may go as low as 4.4V. The inrush current (when charging reservoir and supply decoupling capacitors) is limited by the USB specification (see USB specification v1.1, Section 7.2.4.1). Some applications may require soft start circuitry to limit inrush current if more than 10pF is present between VBUS and GND.

The 5V VBUS line emerging from a PC is often electrically noisy. As well as regulation down to 3.3V and 1.8V, applications should include careful filtering of the 5V line to attenuate noise that is above the voltage regulator bandwidth. Excessive noise on the 1.8V supply to the analogue supply pins of WT module will result in reduced receive sensitivity and a distorted RF transmit signal.

Schematics

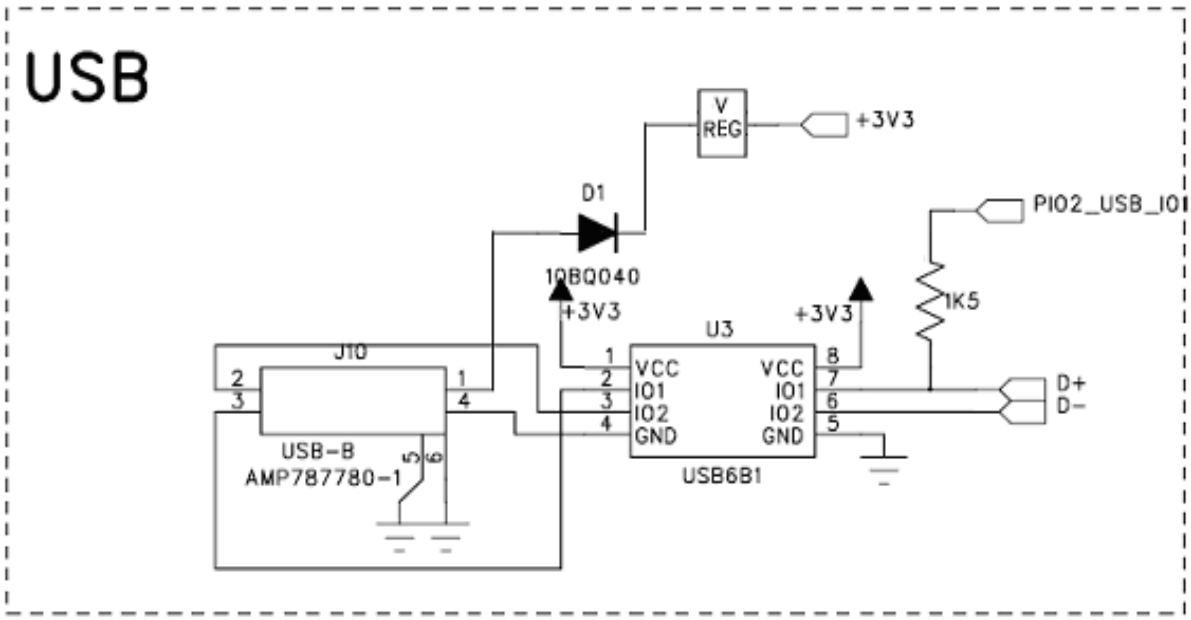

Figure 2: Bus powered USB design

3.1 Parameters

There are a set of parameters that need to be configured for bus powered USB design. Notice that you may also need to change other parameters than the ones described here. Please see Appendix I for all USB related parameters.

3.1.1 PSKEY_HOST_INTERFACE

This key defines the physical connection that is used to pass information to and from the host.

Value: USB link

3.1.2 PSKEY_VM_DISABLE

If TRUE, the VM will not be started when the chip boots. If FALSE, VM operation is normal.

Value: True

3.1.3 PSKEY_ONCHIP_HCI_CLIENT

This key controls the routing of HCI traffic. For a vanilla 'HCI' firmware build this key should be set to FALSE (0). For a firmware build that incorporates upper stack layers, this key should be set to TRUE (1).

Value: 0

3.1.4 PSKEY_USB_PIO_PULLUP

The PIO line used to control the pull-up resistor on the USB D+ line. The absence of this key in the PS indicates that this feature is not in use.

On WT modules, there are 6 PIO lines, 2 to 7. However, an internal pull-up can be used by setting this key to 16.

Value: 16

3.1.5 PSKEY_USB_ATTRIBUTES_POWER

This bit is set if the device is self powered. This bit is described in table 9-8 in USB spec v1.1 - it corresponds to D6 in Offset 7.

Value: No

3.1.6 PSKEY_USB_MAX_POWER

This is the MaxPower field of the local USB device, as defined in Table 9.8 of version 1.1 of the USB specification.

For Class 1 Bluetooth applications, the USB power descriptor should be altered to reflect the amount of power required. This is accomplished by setting the PS key PSKEY_USB_MAX_POWER (0x2c6). This is higher than for a Class 2 application due to the extra current drawn by the Transmit RF PA. WT11 for example can draw 200mA.

Value: Based on design

4. SELF POWERED MODE

In self powered mode, the circuit is powered from its own power supply and not from the VBUS (5V) line of the USB cable. It draws only a small leakage current (below 0.5mA) from VBUS on the USB cable. This is the easier mode for which to design for, as the design is not limited by the power that can be drawn from the USB hub or root port. However, it requires that VBUS be connected to WT modules via a resistor network (Rvb1 and Rvb2), so WT modules can detect when VBUS is powered up. WT modules will not pull USB_DP high when VBUS is off.

Self powered USB designs (powered from a battery or PSU) must ensure that a PIO line is allocated for USB pull-up purposes. A 1.5K +/-5% pull-up resistor between USB_DP and the selected PIO line should be fitted to the design. Failure to fit this resistor may result in the design failing to be USB compliant in self powered mode. The internal pull-up in WT modules is only suitable for bus powered USB devices i.e. dongles.

The terminal marked USB_ON can be any free PIO pin. The PIO pin selected must be registered by setting PSKEY_USB_PIO_VBUS to the corresponding pin number.

In self powered mode PSKEY_USB_PIO_PULLUP must be set to match with the PIO selected.

Note:

USB_ON is shared with WTPIO terminals (PIO2-PIO7).

4.1 Schematics

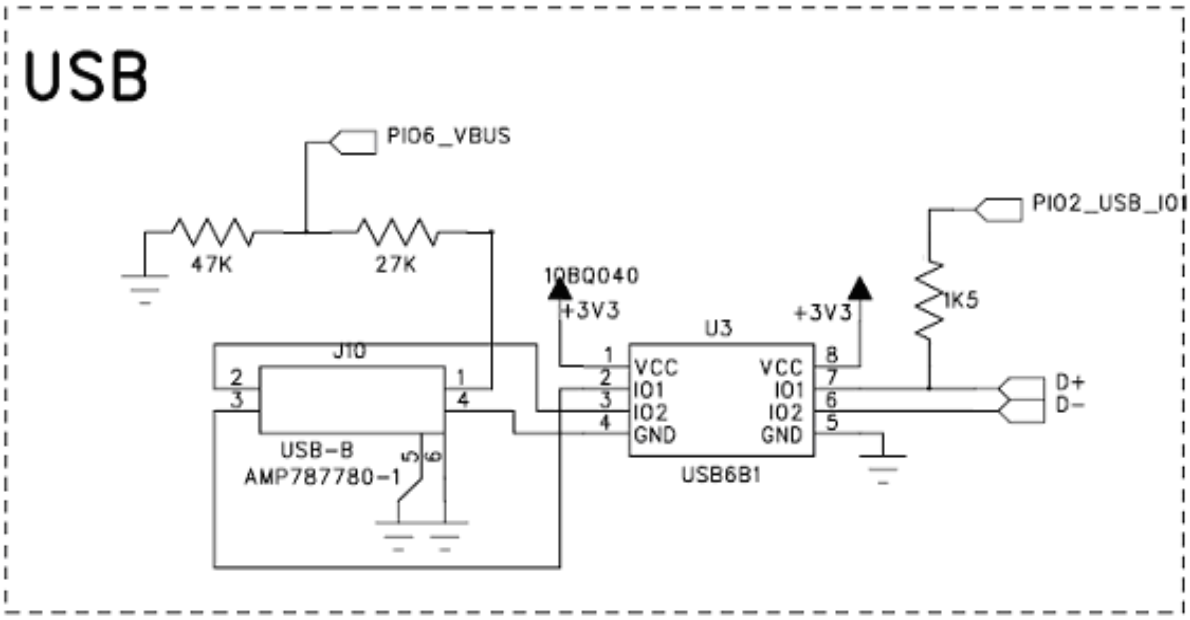

Figure 3: Self powered USB design

4.2 Parameters in self powered mode

There are a set of parameters that need to be configured for self powered USB mode. Notice that you may also need to change other parameters than the ones described here. Please see Appendix I for all USB related parameters.

4.2.1 PSKEY_HOST_INTERFACE

This key defines the physical connection that is used to pass information to and from the host.

Value: USB link

4.2.2 PSKEY_VM_DISABLE

If TRUE, the VM will not be started when the chip boots. If FALSE, VM operation is normal.

Value: True

4.2.3 PSKEY_ONCHIP_HCI_CLIENT

This key controls the routing of HCI traffic. For a vanilla 'HCI' firmware build this key should be set to FALSE (0). For a firmware build that incorporates upper stack layers, this key should be set to TRUE (1).

Value: 0

4.2.4 PSKEY_USB_PIO_VBUS

The PIO line used for USB VBUS detection. This must be one of the 6 available PIO pins (2 to 7). The absence of this key indicates that this feature is not in use.

This value is only used if the chip is presenting its USB interface.

Value: Based on design

4.2.5 PSKEY_USB_PIO_PULLUP

The PIO line used to control the pull-up resistor on the USB D+ line. The absence of this key in the PS indicates that this feature is not in use.

On WT modules, there are 6 PIO lines, 2 to 7.

Value: Based on design

4.2.6 PSKEY_USB_ATTRIBUTES_POWER

This bit is set if the device is self powered. This bit is described in table 9-8 in USB spec v1.1 - it corresponds to D6 in Offset 7.

Value: Yes

4.2.7 PSKEY_USB_MAX_POWER

This is the MaxPower field of the local USB device, as defined in Table 9.8 of version 1.1 of the USB specification.

For Class 1 Bluetooth applications, the USB power descriptor should be altered to reflect the amount of power required. This is accomplished by setting the PS key PSKEY_USB_MAX_POWER (0x2c6). This is higher than for a Class 2 application due to the extra current drawn by the Transmit RF PA. WT11 for example can draw 200mA.

Value: Based on design

5. SOFTWARE

In the previous chapters the USB and hardware specific issues were discussed. This chapter on the other hand briefly describes the software related issues.

With USB there are basically two software approaches.

1. WT modules run HCI firmware
2. WT modules run VM firmware

iWRAP firmware does not support USB at the moment. If VM firmware is required a special firmware needs to be developed using CSR BlueLab software development kit.

5.1 HCI Firmware

WT modules are supplied with Bluetooth v2.0 + EDR compliant stack firmware, which is run on the internal RISC microcontroller.

The software architecture allows Bluetooth processing and the application program to be shared in different ways between the internal RISC microcontroller and an external host processor (if any). The upper layers of the Bluetooth stack (above HCI) can be run either on-chip or on the host processor.

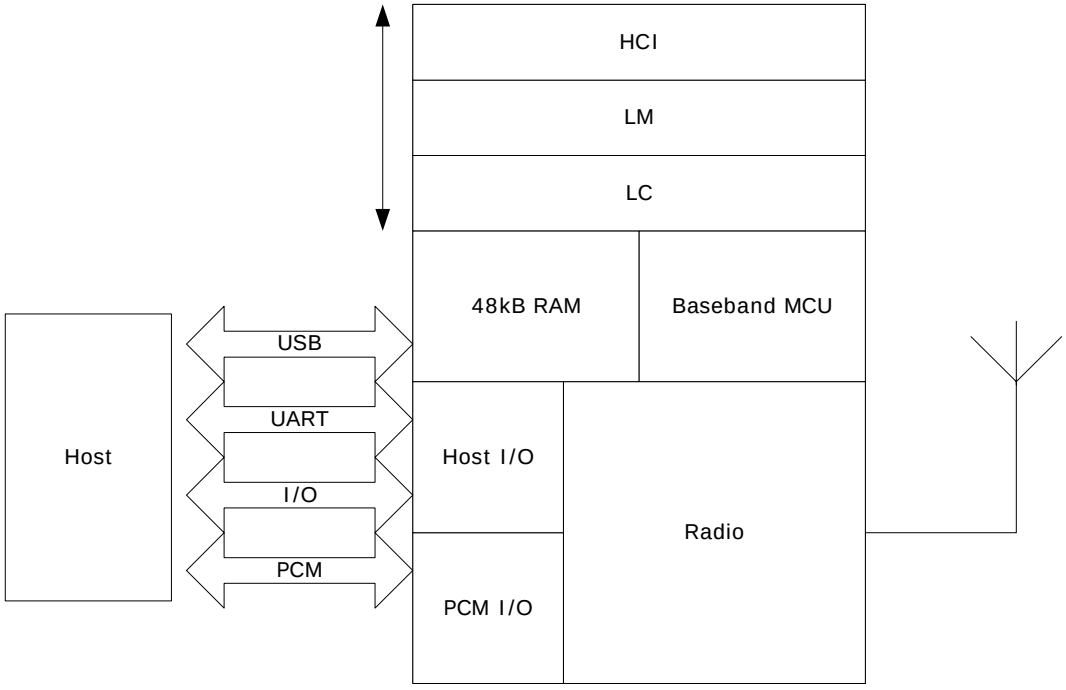

Figure 4: HCI Stack

In the implementation shown in figure above the internal processor runs the Bluetooth stack up to the Host Controller Interface (HCI). The Host processor must provide all upper layers including the application.

5.2 Upper layers of the Bluetooth stack

With the HCI approach the upper layers of the Bluetooth protocol stack as well the Bluetooth profiles need to be run at the host system. Bluegiga Technologies does not

provide the upper software, but it's available through third party software.

5.3 VM Stack

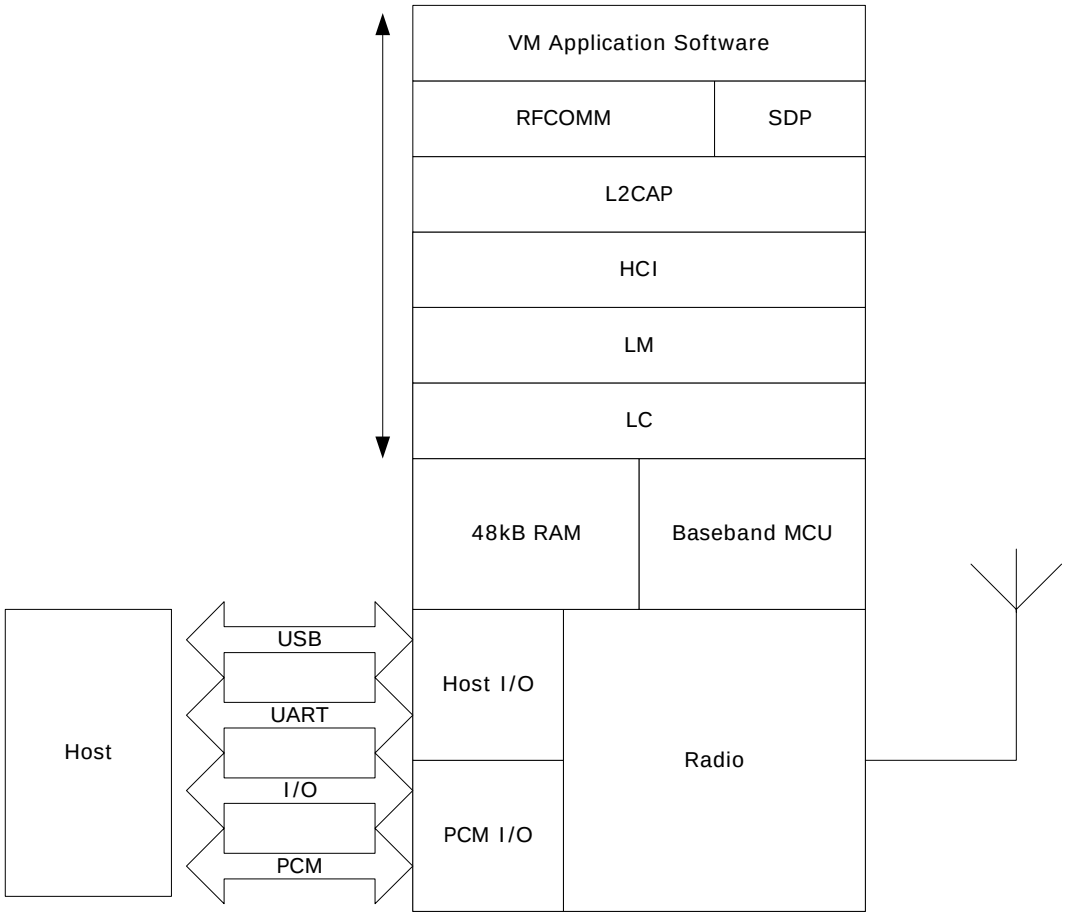

Figure 5: VM Stack

In figure above, this version of the stack firmware shown requires no host processor (but can use a host processor for debugging etc.). All software layers, including application software, run on the internal RISC processor in a protected user software execution environment known as a Virtual Machine (VM).

The user may write custom application code to run on the BlueCore VM using BlueLab™ software development kit (SDK) supplied with the Casira development kit, available separately from Bluegiga or directly form CSR. This code will then execute alongside the main firmware. The user is able to make calls to the firmware for various operations.

The execution environment is structured so the user application does not adversely affect the main software routines, thus ensuring that the Bluetooth stack software component does not need re-qualification when the application is changed.

Using the VM and the BlueLab SDK the user is able to develop applications such as a cordless headset or other profiles without the requirement of a host controller. BlueLab is supplied with example code including a full implementation of the headset profile.

5.4 USB Driver

A USB Bluetooth device driver is required to provide a software interface between WT modules and Bluetooth software running on the host computer. Suitable drivers are

available from www.bluegiga.com/techforum/.

6. DEFAULT PARAMETERS

WT12/WT11–HCI-USB firmware has the following PS-keys. The keys are targeted to designs equal to WT12/WT11 Evaluation Kit.

- Host Interface: USB link
- VM disable: True
- HCI traffic routed internally: 0000
- Initial device boot mode: 0003
- USB D+ pull-up PIO line: 16
- USB product identifier: 0001
- USB responds to wake up: disabled
- USB bus powered: yes
- USB vendor identifier: 0a12

7. CONTACT INFORMATION

Sales: sales@bluegiga.com

Technical support: support@bluegiga.com
http://www.bluegiga.com/techforum/

Orders: orders@bluegiga.com

Head Office / Finland

Phone: +358-9-4355 060
Fax: +358-9-4355 0660

Street Address:
Sinikalliontie 11
02630 ESPOO
FINLAND

Postal address:
P.O. BOX 120
02631 ESPOO, FINLAND

Sales Office / USA
Phone: (781) 556-1039

Bluegiga Technologies, Inc.
99 Derby Street, Suite 200
Hingham, MA 02043

8. APPENDIX I: USB PARAMETERS

Key Name	Key Number	Type	Default Setting
PSKEY_USB_VERSION	0x02bc	uint16	0x0200
The version of the USB spec supported. The value is in BCD, so 0x0200 presents as version "2.0". This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.			

Key Name	Key Number	Type	Default Setting
PSKEY_USB_DEVICE_CLASS_CODES	0x02bd	usbclass	0xe0, 0x01, 0x01
Three bytes giving the class information for the "Standard Device Descriptor" of this USB device, as defined in Table 9.7 of version 1.1 of the USB specification. Type usbclass is held as a uint16[3]. The three values are held in the lower byte of each of the array in the order { class, subclass, protocol }. Default value maps to: { WIRELESS_CONTROLLER, RF_CONTROLLER, BLUETOOTH_PROGRAMMING }. This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.			

Key Name	Key Number	Type	Default Setting
PSKEY_USB_VENDOR_ID	0x02be	uint16	0x0a12

The idVendor field of the local USB device, as defined in Table 9.7 of version 1.1 of the USB specification. This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.

Key Name	**Key Number**	**Type**	**Default Setting**
PSKEY_USB_PRODUCT_ID	0x02bf	uint16	1
The idProduct field of the local USB device, as defined in Table 9.7 of version 1.1 of the USB specification. This value applies when the local device is acting as a Bluetooth device. This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.			

Key Name	**Key Number**	**Type**	**Default Setting**
PSKEY_USB_MANUF_STRING	0x02c1	unicodestring	none
The USB manufacturer string, as described in table 9.7 of the USB specification version 1.1. If no value is stored under this key then there is no such string. This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.			

Key Name	**Key Number**	**Type**	**Default Setting**
PSKEY_USB_PRODUCT_STRING	0x02c2	unicodestring	none

The USB iProduct string, as described in table 9.7 of the USB specification version 1.1. If no value is stored under this key then there is no such string. This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.

Key Name	**Key Number**	**Type**	**Default Setting**
PSKEY_USB_SERIAL_NUMBER_STRING	0x02c3	unicodestring	none
The USB serial number string, as described in table 9.7 of the USB specification version 1.1. If no value is stored under this key then there is no such string. This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.			

Key Name	**Key Number**	**Type**	**Default Setting**
PSKEY_USB_CONFIG_STRING	0x02c4	unicodestring	none
The USB configuration string, as described in table 9.8 of the USB specification version 1.1. If no value is stored under this key then there is no such string. This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.			

Key Name	**Key Number**	**Type**	**Default Setting**
PSKEY_USB_ATTRIBUTES	0x02c5	uint8	0xc0

The bmAttributes of the Standard Configuration Descriptor, described in Table 9.8 of USB specification, version 1.1. The default value (0xc0) means that the device is self-powered and that remote wakeup is not supported. This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.

Key Name	**Key Number**	**Type**	**Default Setting**
PSKEY_USB_MAX_POWER	0x02c6	uint16	0
The MaxPower field of the local USB device, as defined in Table 9.8 of version 1.1 of the USB specification. This value is given in units of 2 milliamps (sic). This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.			

Key Name	**Key Number**	**Type**	**Default Setting**
PSKEY_USB_BT_IF_CLASS_CODES	0x02c7	usbclass	0xe0, 0x01, 0x01

Three bytes giving the class information for the USB "Interface Descriptor" (as defined in Table 9.9 of version 1.1 of the USB specification) which contains the USB control, BT HCI event and BT ACL endpoints. The class codes for the BT interface containing the SCO endpoints are given by PSKEY_USB_BT_SCO_IF_CLASS_CODES.

Type usbclass is held as a uint16[3]. The three values are held in the lower byte of each of the array in the order

{ class, subclass, protocol }.

Default value maps to:

{ WIRELESS_CONTROLLER, RF_CONTROLLER, BLUETOOTH_PROGRAMMING }.

This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.

Key Name	**Key Number**	**Type**	**Default Setting**
PSKEY_USB_LANGID	0x02c9	uint16	0x0409

Defines the languages supported by the USB interface. The languages are used in USB identification strings as described in section 9.6.5 of version 1.1 of the USB specification. (Microsoft type "LANGID", used by the USB spec.)

The default value of 0x0409 maps to:

Primary language id ENGLISH (1),

Secondary language id ENGLISH_US (9).

Value is 0 if no language strings are supported.

This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.

Key Name	**Key Number**	**Type**	**Default Setting**
PSKEY_USB_DFU_CLASS_CODES	0x02ca	usbclass	0xfe, 0x01, 0x00

Three bytes giving the class information for an "Interface Descriptor" as defined in Table 9.9 of version 1.1 of the USB specification. This key describes the DFU (Device Firmware Upgrade) interface of this USB device.

Type usbclass is held as a uint16[3]. The three values are held in the lower byte of each of the array in the order

{ class, subclass, protocol }.

The default value maps to { APPLICATION_SPECIFIC_CLASS, DFU, NO_PROTOCOL }.

This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.

Key Name	**Key Number**	**Type**	**Default Setting**
PSKEY_USB_DFU_PRODUCT_ID	0x02cb	uint16	0xffff

The idProduct field of the local USB device, as defined in Table 9.7 of version 1.1 of the USB specification. This value applies when the local device is acting as a DFU (Device Firmware Upgrade) device.

This must be different from the value held under PSKEY_USB_PRODUCT_ID.

This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.

Key Name	**Key Number**	**Type**	**Default Setting**
PSKEY_USB_PIO_DETACH	0x02ce	uint16	none

This PS key sets the PIO line used for USB detach/attach signaling. This must be one of the 2 available interrupt input pins (4 or 5). The absence of this key indicates that this feature is not in use.

This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.

Key Name	Key Number	Type	Default Setting
PSKEY_USB_PIO_WAKEUP	0x02cf	uint16	none
The PIO line used for USB wakeup signaling in detach mode. The absence of this key indicates that this feature is not in use. This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.			

Key Name	Key Number	Type	Default Setting
PSKEY_USB_PIO_PULLUP	0x02d0	uint16	16
The PIO line used to control the pull-up resistor on the USB D+ line. The absence of this key in the persistent storage indicates that this feature is not in use. This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE. On BlueCore, there are 16 PIO lines, 0 to 15 (depending on package type), but 12 to 15 are not available for the USB pull-up. However, an internal pull-up can be used by setting this key to 16.			

Key Name	Key Number	Type	Default Setting
PSKEY_USB_PIO_VBUS	0x02d1	uint16	none

The PIO line used for USB VBUS detection. This must be one of the 2 available interrupt input pins (4 or 5). The absence of this key indicates that this feature is not in use. This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.

Key Name	**Key Number**	**Type**	**Default Setting**
PSKEY_USB_PIO_WAKE_TIMEOUT	0x02d2	uint16	0
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 is 0, the signal is asserted indefinitely (or until the host de-asserts detach). This key is useful 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 could 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. This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.			

Key Name	**Key Number**	**Type**	**Default Setting**
PSKEY_USB_PIO_RESUME	0x02d3	uint16	none

The PIO line used to signal that the USB host wakeup from suspend. The absence of this key indicates that this feature is not in use.

PIO resume is used in place of the bus resume signal for hosts that are unable to respond to the bus signal during suspend e.g. PDAs that power-down the root hub in 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 this PS key is sufficient to enable this feature; no notice is taken of the remote wakeup setting of PSKEY_USB_ATTRIBUTES nor of whether the host has enabled remote wakeup.

PSKEY_USB_PIO_RESUME is mutually exclusive with PSKEY_USB_PIO_WAKEUP - both can be enabled simultaneously and assigned to the same PIO pin.

This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.

Key Name	**Key Number**	**Type**	**Default Setting**
PSKEY_USB_BT_SCO_IF_CLASS_CODES	0x02d4	usbclass	0xe0, 0x01, 0x01

Three bytes giving the class information for the USB "Interface Descriptor" (as defined in Table 9.9 of version 1.1 of the USB specification) which contains the SCO endpoints.

This PS key enables different class codes to be used for the BT SCO interface from the main BT interface (which contains the control, event and ACL endpoints and whose class codes are given by PSKEY_USB_BT_IF_CLASS_CODES). The advantage of this is that the host OS can load different drivers to communicate with each of the interfaces. This is expected to be of use for Microsoft's XP OS in which the in-built BT USB driver will not give access to the SCO endpoints.

Type usbclass is held as a uint16[3]. The three values are held in the lower byte of each of the array in the order:

{ class, subclass, protocol }.

Default value maps to:
{ WIRELESS_CONTROLLER, RF_CONTROLLER, BLUETOOTH_PROGRAMMING }.

This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.

Key Name	Key Number	Type	Default Setting
PSKEY_USB_SUSPEND_PIO_LEVEL	0x02d5	uint16	0

This description covers PSKEY_USB_SUSPEND_PIO_LEVEL, PSKEY_USB_SUSPEND_PIO_DIR, PSKEY_USB_SUSPEND_PIO_MASK.

To ensure that the limit on power drawn in suspend mode for a bus-powered device is met, BlueCore usually sets all PIO lines to low. However, this may not always be correct and this set of PS keys allows 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 that will be left alone.

For each bit that is set to 1 in PSKEY_SUB_SUSPEND_PIO_MASK, a 0 (1) for the corresponding bit in PSKEY_USB_SUSPEND_PIO_LEVEL indicates that the line should be set low (high), and a 0 (1) for the corresponding bit in PSKEY_USB_SUSPEND_PIO_DIR indicates that the line will be set for input (output). Note that from BlueCore2 on 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. Therefore for BlueCore 1, PSKEY_USB_SUSPEND_PIO_DIR is not used and any line to be driven is set for output.

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

The names of these keys are misleading as the same facilities are used to handle BCCMDVARID_WARM_HALT and BCCMDVARID_COLD_HALT, regardless of the host transport.

Key Name	Key Number	Type	Default Setting
PSKEY_USB_SUSPEND_PIO_DIR	0x02d6	uint16	0

See PSKEY_USB_SUSPEND_PIO_LEVEL.

Key Name	Key Number	Type	Default Setting
PSKEY_USB_SUSPEND_PIO_MASK	0x02d7	uint16	0xffff
See PSKEY_USB_SUSPEND_PIO_LEVEL.			

Key Name	Key Number	Type	Default Setting
PSKEY_USB_ENDPOINT_0_MAX_PACKET_SIZE	0x02d8	uint8	64
The bMaxPacketSize0 field of the USB Standard Device Descriptor, as defined in Table 9.7 of version 1.1 of the USB specification. Only values 8, 16, 32 and 64 are valid for this field. This value is only used if the chip is presenting its USB interface. See the description of PSKEY_HOST_INTERFACE.			

Key Name	Key Number	Type	Default Setting
PSKEY_USB_CONFIG	0x02d9	uint16	0x30
This key modifies the behavior of the USB interface code. It should only be configured on advice from CSR.			