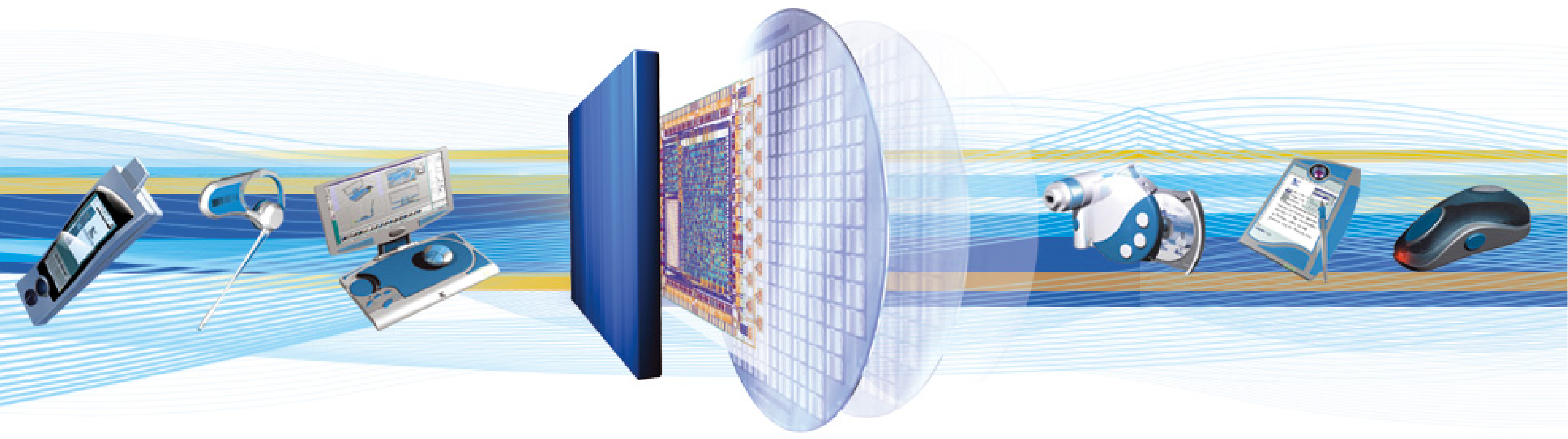
BlueCore™

Booting BlueCore ROM

August 2005



CSR

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

Registered in England 3665875

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

Contents

1 Introduction

All HCI firmware builds up to HCIStack1.1v16.x have been held in flash memory. Part of the flash is used to form a “Persistent Store”, holding the firmware’s set of configuration data, e.g., its Bluetooth address.

From HCIStack1.1v17.x, the firmware can be in ROM, and the **BlueCore™** chip may have no writable/persistent storage in which to hold its configuration data. The baseband needs configuration data to run and the host must provide it when the chip is initialised. This document describes this mechanism.

This document also gives brief details of the firmware’s mechanism for working in systems where it does not know its clock frequency.

2 Configuration Data Storage Options

The firmware holds its configuration data in the Persistent Store, as described in the Persistent Store Commands section of [BCCMDS]. In all HCI builds up to HCIStack1.1v16.x this store is located in part of the same flash memory device used to hold the firmware.

In later builds there are three options for holding Persistent Store data:

1. **Flash memory**: These build variants hold their configuration data in flash memory pages, exactly as with HCIStack1.1v16.x builds.
2. **RAM**: If a BlueCore chip holds its program in ROM, and if there is no writable storage, then the host is required to pass configuration data to BlueCore during initialisation. The data is held in RAM, so is lost when the chip's power is removed. This approach requires a special initialisation sequence and is the primary subject of this document.
3. **EEPROM**: A BlueCore chip holding its program in ROM can optionally be connected to an external EEPROM via an I^{2}C bus. The EEPROM is used to hold configuration information. Conceptually this is the same as holding configuration data in flash memory, though EEPROM's intrinsically slow access time presents its own problems.

3 Booting BlueCore ROM

Where a BlueCore ROM device has no flash or EEPROM storage, device-specific configurations (e.g., the device's Bluetooth address) must be loaded into the chip's RAM at boot.

A dual-boot sequence must be used:

1. Hardware reset.
2. Load configuration values to baseband RAM.
3. Reboot the baseband using a "warm reset" (to maintain the configuration values).
4. Use the radio for Bluetooth operations.

3.1 BCSP Boot Sequence

A normal BCSP boot sequence for a ROM part is as follows.

1. The chip is reset by one of the following methods: powering up, toggling the hardware reset pin, using the BCCMD "cold reset" command, or a watchdog reset.
2. The firmware senses whether an EEPROM device is accessible by sensing on PIO[6:8]. Assume the EEPROM device is not present.
3. The chip's digital circuitry (including the CPU) is normally driven from a 16MHz internal clock. This is taken from a PLL that is normally locked to an external reference frequency (a system clock or a crystal). However, when the chip first boots, the PLL is not locked to the reference, so it free runs at about 4MHz. This allows the CPU to run, but rather slower than normal.
4. The firmware checks the value of PSKEY_ANA_FREQ and finds that no value is stored (the default). The firmware takes this to mean that it does not know the frequency of the system clock. Consequently, it tries to lock the chip's PLL to the system clock input by trying a sequence of possible frequencies in turn.

   The digital circuitry must not be run faster than 16MHz, so the sequence of (divide ratios for) frequencies loaded into the PLL is tried in reverse order, starting at 39MHz and working down.

   All that matters at this stage is that the PLL locks to the system clock; it is not necessary for the firmware to know the actual system clock frequency. For example, if the system clock is 16MHz, it is acceptable for the PLL to be set (and to lock) as if the system clock is 18MHz. This means the digital circuitry will be running at roughly 14MHz.

   The initialisation code also uses the fact that no value is stored under PSKEY_ANA_FREQ to decide not to initialise the radio. Consequently, any attempt by the host to use the radio in this state will fail.
5. If the PS Key store holds a required baud rate of zero, then the support to adapt to the host's baud rate is invoked, described in [AUTOBAUD]. Note that this blocks the initialisation sequence, so operations such as using `pstool.exe` to change PS Keys' values over SPI will not run until the UART has been configured.
6. The host transport performs BCSP Link Establishment.
7. The firmware emits an HCI Command_Status event with opcode NOP (0). This is the normal way that CSR firmware tells its host that it is ready for operation.
8. The host writes to a set of PS Keys; the firmware stores these in RAM (in psram). This set must include the device's Bluetooth address and the system clock frequency (PSKEY_ANA_FREQ), but it will probably include other values, e.g., the radio's power table, a constraint on the maximum effective encryption length and the actual baud rate. section 4 covers PS Key settings.
9. The host may choose to set the values of the firmware's "panic argument" and "fault argument" to zero at this point. The sections on the "Panic Arg" and "Fault Arg" BCCMD commands in [BCCMDS] describe this functionality.
10. The host commands the firmware to perform a "warm reboot" (described in [BCCMDS]). This reboots the chip, but values held in psram are preserved through the boot. Thus, the device starts up knowing its required Bluetooth address, clock frequency, etc. (A warm reset is called a warm reboot in other documentation.)

11. The firmware configures the PLL (to match the real frequency of the system clock) according to the contents of PSKEY_ANA_FREQ.
12. The initialisation code uses the value stored in PSKEY_ANA_FREQ to build the radio's trim table, allowing the radio to operate.
13. The PS Keys may have been set to change the UART's baud rate, the host transport, the mapping of PIO pins, etc. The firmware will configure itself accordingly. For example, the new PS Keys' settings may cause BCSP to run at 921.6kbaud after the warm reboot. (If the PS Key store still does not define the host's baud rate, the [AUTOBAUD] mechanism is invoked again.)
14. Assuming the host transport is still BCSP, the Link Establishment procedure is performed again.
15. The firmware emits a second HCI Command_Status NOP.

The host can now perform Bluetooth operations.

3.2 H4/H4DS Boot Sequence

Booting the chip to use H4/H4DS is conceptually the same as for BCSP.

Care must be taken when the warm reset is performed. With the current firmware, the host will not receive a reply to the Warm_Reset BCCMD command; the chip is rebooted almost immediately after it receives the command. If H4/H4DS is to use a different baud rate after the warm reboot, the normal boot-time NOP will be emitted at the new rate. The host must thus switch to the new baud rate briskly after commanding the warm reboot, or the NOP will be lost and the transport will lose synchronisation. (The second boot-time NOP could be suppressed by setting PSKEY_HCI_NOP_DISABLE.)

3.3 USB Boot Sequence

Support for booting USB devices from ROM has varied in different firmware releases. For example, firmware build HCIStack1.1v17.6.3 in BlueCore ROM only supports USB (invoked by PIO values read while booting the firmware) with a 16MHz crystal and PS Key values stored in EEPROM. This removes the need for the dual-boot approach.

The user is advised to check the Software Release Note for the firmware build in question to check what USB support is included.

4 PS Key Settings

When using a ROM-only device, the host must write a set of PS Key values to the firmware's RAM during booting.

Some values must be set per device (e.g., the Bluetooth address), and some per (radio) design (e.g., the power table). This section lists PS Keys that must be set in each of these cases. It also lists keys that may need to be set per device and per design. These PS Key lists are not complete: designs may need to configure extra PS Keys.

4.1 Required PS Key Settings (Per Device)

The host *must* set the following PS Key during booting. This PS Key's value is unique for each Bluetooth device.

- PSKEY_BDADDR: Bluetooth does not work too well if all devices have the same Bluetooth address.

4.2 Required PS Key Settings (Per Product Design)

The host *must* set the following PS Keys during booting. This PS Key's value is unique for each Bluetooth (radio) design.

- PSKEY_LC_POWER_TABLE: Each radio design has different RF characteristics, so each product design must have a calibrated power table.
- PSKEY_ANA_FREQ to tell the firmware the frequency of its system clock.

4.3 Possible PS Key Settings (Per Product Design)

The host *may* also need to set the following PS Keys during booting.

- PSKEY_LC_DEFAULT_TX_POWER: The Bluetooth power class determines the transmit power to be used for Page, Page-Scan, Inquiry and Inquiry-Scan. The transmit power is also used on fresh ACL connections (before LMP power control comes into play).
- PSKEY_LC_MAX_TX_POWER: The Bluetooth power class determines the maximum transmit power allowed.
- PSKEY_ENC_KEY_LMAX: This sets the maximum effective encryption key length. Most CSR ROM firmware defaults to using 128-bit encryption, but some governments require the use of weaker protection.
- PSKEY_HOSTIO_UART_RESET_TIMEOUT: If set, the UART provokes a hardware reset if it detects a long break condition.
- PSKEY_PCM_CONFIG32 and PSKEY_PCM_FORMAT: These configure the PCM port's settings.
- PSKEY_HOSTIO_MAP_SCO_PCM: Routing SCO data over HCI or through the PCM port.
- PSKEY_WD_TIMEOUT and PSKEY_WD_PERIOD: These configure the chip's watchdog.
- PSKEY_TXRX_PIO_CONTROL: This configures how PIO pins can drive an external PA/LNA.
- PSKEY_HOSTIO_UART_PS_BLOCK: This includes a field that sets the baud rate.

4.4 Host PS Key Support

Section 4.3 (listing PS Keys that may need to be changed for each design), is almost certain to be incomplete. Currently, there are approximately 400 PS Keys, and experience has shown that it is very valuable to be able to change PS Keys' settings at any stage of development. For example, when a ROM chip is assessed, it is highly likely the radio will be found to benefit from some small PS Key value changes.

To support this, it can be useful for the host to run something like the following (almost) C code while the chip is being initialised:

```
static PSDATA psdata[] = {
/* {pskey,                 len,  *data}, */
/* uint16,              uint16,  uint16[] */

   {PSKEY_BDADDR,          6,    {0x00, 0x02, 0x5b, 0x01, 0x02, 0x03}},
   {PSKEY_LC_MAX_TX_POWER, 1,    {0x04}},
   {PSKEY_ENC_KEY_LMAX,    1,    {0x07},

   {PSKEY_NULL,            0,    NULL}}
   };
…
for(p=psdata; p->pskey != PSKEY_NULL; p++)
   ps_write(p->pskey, p->len, p->data);
```

(PSKEY_NULL is zero; it is guaranteed that no PS Key has this value.)

Ideally, the contents of `psdata[]` should be taken from some form of configuration file, so that PS Key settings can be changed just by altering the file's contents.

Separating the per-module and per-design settings may simplify product manufacture.

Document References

Document ID	Document Title	CSR Reference
[AUTOBAUD]	UART Baud Rate Adaptation	bcore-me-019P
[BCCMDS]	BCCMD Commands	bcore-sp-005P
[BT2.0]	Specification of the Bluetooth System, Version 2.0 + EDR, Core Package, 4 November 2004	n/a
[H4DS]	H4DS Protocol	bcore-sp-013P

Terms and Definitions

ACL	Asynchronous Connection-Less
BCCMD	BlueCore Command
BCSP	BlueCore Serial Protocol
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
CDMA	Code Division Multiple Access
CSR	Cambridge Silicon Radio
EEPROM	Electrically Erasable Programmable Read Only Memory
H4	UART-based HCI transport, defined in [BT2.0]
H4DS	H4 Deep Sleep – an extension to the H4 protocol, defined by [H4DS]
HCI	Host Controller Interface
I^2C	Inter Integrated Circuit
LMP	Link Manager Protocol
LNA	Low Noise Amplifier
NOP	No Operation (code)
PA	Power Amplifier
PCM	Pulse Code Modulation
Persistent Store	Storage of BlueCore's configuration values in non-volatile memory
PIO	Programmable Input Output
PLL	Phase Lock Loop
PS Key	Persistent Store Key
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VM	Virtual Machine

Document History

Revision	Date	History
a	02 Jun 03	Original internal version of this document.
b	29 Aug 03	Initial publication of this document.
c	01 Mar 04	Added description of operation with an unknown system clock frequency, introduced in HCIStack1.2v17.6.3. Rewrote description for operation with USB.
d	02 Aug 05	Changes to BCSP boot sequence, more information on USB boot sequence, corrected typos, minor formatting updates.

Booting **BlueCore™** ROM

bcore-me-014Pd

August 2005