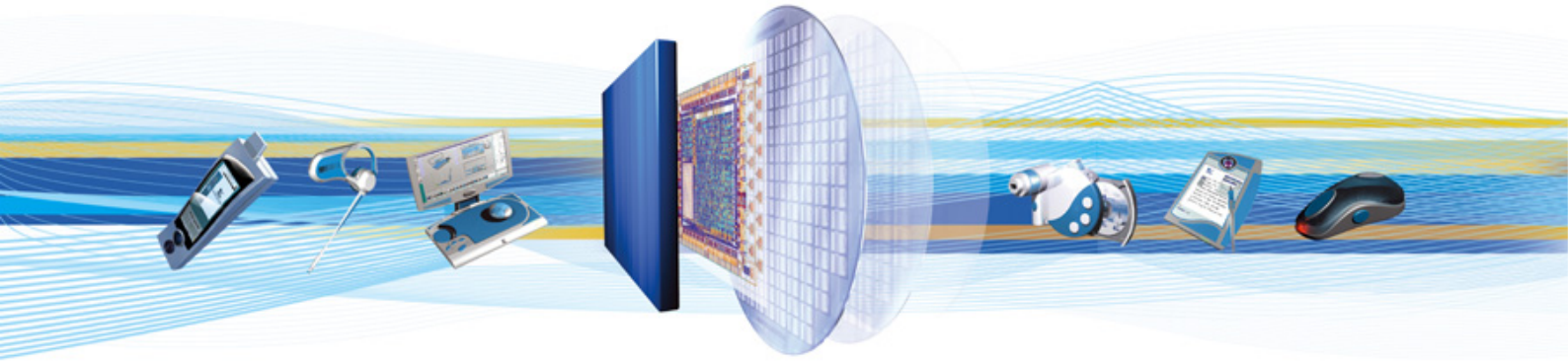
CSR Synergy Framework 3.1.0

BCCMD – Bluecore Command

API Description

August 2011



Cambridge Silicon Radio Limited

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

Registered in England and Wales 3665875

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

Contents

List of Figures

List of Tables

1 Introduction

1.1 Introduction and Scope

This document describes the functionality and message interface provided by the BC Command Specification in CSR Synergy Framework, henceforward called BCCMD. The BCCMD protocol interfaces to a command interpreter on a CSR Bluetooth® chip. The command interpreter presents commands that allow monitoring and control of the chip. The command set is not part of the Bluetooth® standard.

For a more thorough description of the basic structure and mechanisms of the protocol, refer to [BCCMD]. Actual commands are described in [BCCMDS].

2 Description

The BCCMD protocol allows an application to monitor and control the BlueCore chip. The basic functionality is based on the model illustrated in Figure 1.

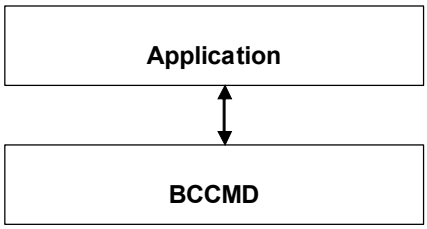


Figure 1: Reference model

The protocol is modelled on a conventional client/server structure. The client (the application) sends a request through the BCCMD module to the server (the chip), the server attempts to service the request and returns the responses to the client.

2.1 Messages

This section describes the BCCMD messages. A message is shaped to fit the XAP processor within the BlueCore chip in which each data address accesses a 16-bit value. This means that the fundamental unit storage in BCCMD messages is a 16-bit unsigned integer, rather than a byte.

As illustrated in Figure 2, the 16-bit format in the BCCMD message must be with the less significant bit first. 8-bit integers must be promoted to a 16-bit integer size before transmission.

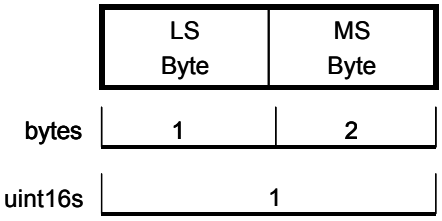


Figure 2: 16-Bit format

32-bit integers have a more awkward byte ordering, as illustrated in Figure 3.

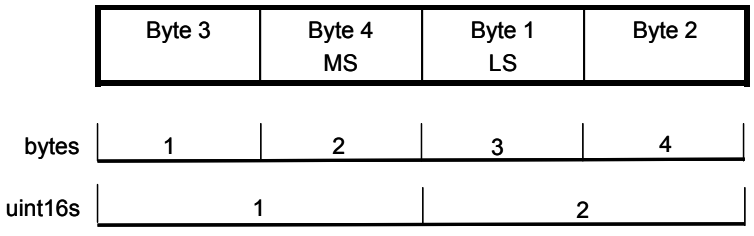


Figure 3: 32-Bit format

The BCCMD messages use a common format, as illustrated in Figure 4.

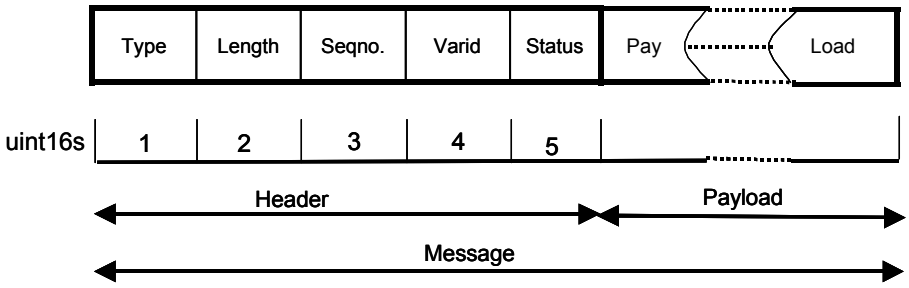


Figure 4: Common format message structure

The whole message must be an even number of bytes in length, and have a minimum size of 18 bytes and the message header's fields are described as follows:

- The **Type** field describes the type of the message. There are only three message types:

Name	Value	Direction
GETREQ	0x0000	Client to server
GETRESP	0x0001	Server to Client
SETREQ	0x0002	Client to server

- The **Length** field describes the total length of the message measured in 16-bit integers. The whole message must be an even number of bytes, so the value of this field is half the length of the message measured in bytes. Note that the *length* field must correspond to the BCCMD_MSG_REQ/CFM *payloadLength* field (mind the different between word size and byte size)
- The client chooses the **Seqno** value for each GETREQ and SETREQ. The server copies this into its response GETRESP messages, allowing the client to pair responses with the original requests
- The **Varid** field holds the identifier of the variable being manipulated by the message. The protocol views the server's resources as a name/value database, where the **Varid** field "names" the database variable being accessed
- The **Status** field gives status information about the transaction. In GETREQ and SETREQ messages, the **Status** field must always be 0x0000 (meaning OK). In GETRESP messages, the **Status** field carries a report on the success of the operation requested by the responding GETREQ or SETREQ message. In [BCCMD] it is defined which values the **Status** may take in a GETRESP
- All messages must include a **Payload** field being at least four uint16s in length. The **Payload** field is an integer number being uint16s long and is sized to fit the varid's value if this does not fit within the four uint16s. The majority of varids can fit their values in the four uint16s **Payload** field. If the four uint16s are insufficient to hold the value, the **Payload** field is enlarged in order to fit. Details of these cases are given in [BCCMD] and [BCCMDS]. Please notice that any part of a **Payload** field that does not have a useful value must be set to zero. The **Payload** can contain any type of value as long as it is packed in uint16s.

3 Interface Description

In this section a series of MSCs will be shown to explain the usage of the BCCMD module. The primitives and the functions available to the application are also described in the subsections of this chapter.

3.1 Transfer BCCMD Messages

To transfer a BCCMD message from the application to the Bluecore chip, four different “send” functions are available depending of the type of information that is send to the Bluecore Chip. The functions all use the same set of primitives to interface with the BCCMD module.

The four send functions are:

1) CsrBccmdWriteReqSend - used for building a SETREQ BCCMD message

2) CsrBccmdReadReqSend – used for building a GETREQ BCCMD message

3) CsrBccmdReadPsValueReqSend – used for building a BCCMD message that will read some PS-keys from the Bluecore

4) CsrBccmdWritePsValueReqSend – used for building a BCCMD message that will write some PS-keys on the Bluecore

The prototypes for these functions and a description of the parameters will be given later - but they all use the same two primitives for carrying the information as shown on Figure 5.

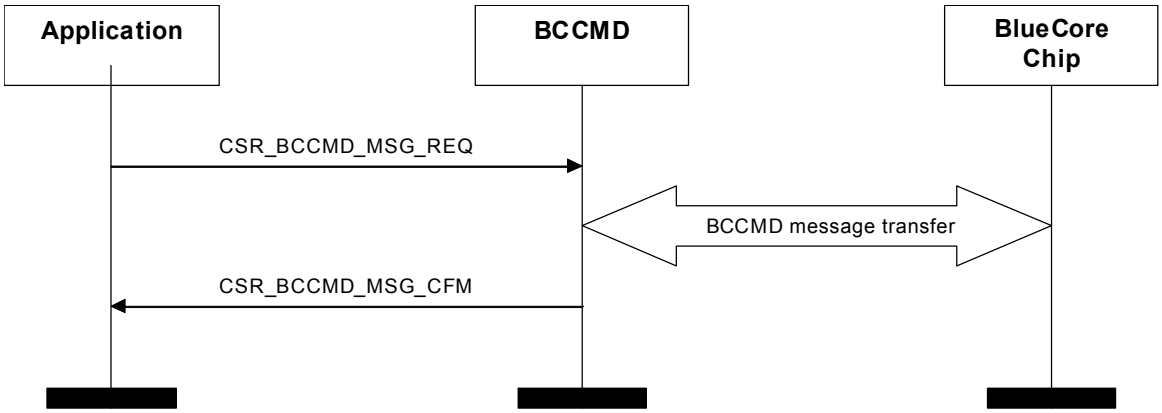


Figure 5: BCCMD message transfer

As seen in Figure 5 the resulting primitive is always the CSR_BCCMD_MSG_CFM, which is described in Table 1.

Type	Parameter	Description
CsrBccmdPrim	type	Signal identity – always CSR_BCCMD_MSG_CFM
CsrUint16	cmdType	BCCMD command type
CsrUint16	seqNo	Sequence number of the BCCMD message
CsrUint16	varId	VarId of the BCCMD message
CsrResult	status	Status information about the BCCMD transaction
CsrUint16	payloadLength	The length of the payload of the BCCMD message
CsrUint8	*payload	Pointer to the payload of the BCCMD message
Note: It is the responsibility of the application to free payload.		

Table 1: Parameters in a CSR_BCCMD_MSG_CFM primitive

If CSR_BLUECORE_ONOFF is defined and the BlueCore enters the deactivating state, any outstanding BCCMD requests will be discarded, and no confirm messages will be sent to the tasks that issued the BCCMD requests. Any BCCMD requests received during the deactivation process will also be discarded. When the BlueCore is in the deactivated state (either before the first activation or after deactivation completes) or in the activating state, BCCMD requests will be queued and processed when the BlueCore enters the active state.

3.2 CsrBccmdWriteReqSend

The CsrBccmdWriteReqSend function is used for building and sendinig a SETREQ bccmd message.

The prototype of the function is:

```
CsrBccmdWriteReqSend(CsrQid phandle, CsrUint16 varId,
                     CsrUint16 seqNo,
                     CsrUint16 payloadLength,
                     CsrUint8       *payload);
```

The arguments to the function are described in Table 2.

Type	Argument	Description
CsrQid	phandle	Identity of the calling process
CsrUint16	varId	Identifier of the variable being manipulated by the message
CsrUint16	seqNo	Value in returned in the confirmed message
CsrUint16	payloadLength	Length of the message payload field measured in 8-bit integers. **Note that the payload length must be an even number**
CsrUint8	*payload	Payload field of the BCCMD message. The payload must in XAP format. E.g. 8-bit integers travel as 16-bit integers, 16-bit integers travel with the less significant byte first

Table 2: Arguments to `CsrBccmdWriteReqSend` function

3.2.1 CsrBccmdReadReqSend

The CsrBccmdReadReqSend function is used for building and sending a GETREQ bccmd message.

The prototype of the function is:

```
CsrBccmdReadReqSend(CsrQid phandle, CsrUint16 varId,
                    CsrUint16 seqNo,
                    CsrUint16 payloadLength,
                    CsrUint8       *payload);
```

The arguments to the function are described in Table 3

Type	Argument	Description
CsrQid	phandle	Identity of the calling process
CsrUint16	varId	Identifier of the variable being manipulated by the message
CsrUint16	seqNo	Value in returned in the confirmed message
CsrUint16	payloadLength	Length of the message payload field measured in 8-bit integers. **Note that the payload length must be an even number**

Type	Argument	Description
CsrUint8	*payload	Payload field of the BCCMD message. The payload must in XAP format. E.g. 8-bit integers travel as 16-bit integers, 16-bit integers travel with the less significant byte first

Table 3: Arguments to `CsrBccmdReadReqSend` function

3.2.2 CsrBccmdReadPsValueReqSend

The CsrBccmdReadPsValueReqSend function is used for building and sending a GETREQ bccmd message with VarId 0x7003 e.g. a persistent store command

The prototype of the function is:

```
CsrBccmdReadPsValueReqSend(CsrQid                phandle,
                           CsrUint16             seqNo,
                           CsrUint16             key,
                           CsrBccmdStoresType    stores,
                           CsrUint16             psValuelength);
```

The arguments to the function are described in Table 4

Type	Argument	Description
CsrQid	phandle	Identity of the calling process
CsrUint16	seqNo	Value in returned in the confirmed message
CsrUint16	key	Identifier of the PS Key of the database element being accessed
CsrBccmdStoresType	stores	Controls the searching of the four component stores that make up the fullPersistent Store. Stores field values are defined in csr_bccmd_prim.h
CsrUint16	psValuelength	The psValuelength of the PS Value field measured in 8-bit integers. Note that the psValuelength must be an even number and it must be set to the maximum length of the data that can be taken from persistent Store in this operation

Table 4: Arguments to `CsrBccmdReadPsValueReqSend` function

3.2.3 CsrBccmdWritePsValueReqSend

The CsrBccmdWritePsValueReqSend function is used for building and sending a SETREQ bccmd message with VarId 0x7003, e.g. a persistent store command

The prototype of the function is:

```
CsrBccmdWritePsValueReqSend(CsrQid                phandle,
                            CsrUint16             seqNo,
                            CsrUint16             key,
                            CsrBccmdStoresType    stores,
```

```
                                    CsrUint16           psValuelength);
```

The arguments to the function are described in Table 5

Type	Argument	Description
CsrQid	phandle	Identity of the calling process
CsrUint16	seqNo	Value in returned in the confirmed message
CsrUint16	key	Identifier of the PS Key of the database element being accessed
CsrBccmdStoresType	stores	Controls the searching of the four component stores that make up the fullPersistent Store. Stores field values are defined in csr_bccmd_prim.h
CsrUint16	psValuelength	The psValuelength of the PS Value field measured in 8-bit integers. Note that the psValuelength must be an even number and it must be set to the maximum length of the data that can be taken from persistent Store in this operation

Table 5: Arguments to `CsrBccmdWritePsValueReqSend` function

4 Document References

Document	Reference
[BCCMD]	BCCMD Protocol, CSR document bcore-sp-002Pc, see http://www.csrsupport.com/document.php?did=404
[BCCMDS]	BCCMD Commands; CSR document bcore-sp-005Pd, see http://www.csrsupport.com/document.php?did=405

Terms and Definitions

BlueCore®	Group term for CSR's range of Bluetooth® chips.
XAP	Microprocessor at the heart of the BlueCore chip.
Varid	VARidble IDentifier.
Persistent Store	Storage of BlueCore's configuration values in non-volatile memory.

Document History

Revision	Date	History
1	19 DEC 08	Ready for first Engineering Release
2	16 JAN 09	Ready for second Engineering Release
3	09 FEB 09	Ready for release 1.0.0
4	26 MAY 09	Ready for release 1.1.0
5	30 NOV 09	Ready for release 2.0.0
6	20 APR 10	Ready for release 2.1.0
7	OCT 10	Ready for release 2.2.0
8	DEC 10	Ready for release 3.0.0
9	Aug 11	Ready for release 3.1.0

TradeMarks, Patents and Licences

Unless otherwise stated, words and logos marked with ™ or ® are trademarks registered or owned by CSR plc or its affiliates. Bluetooth® and the Bluetooth logos are trademarks owned by Bluetooth SIG, Inc. and licensed to CSR. Other products, services and names used in this document may have been trademarked by their respective owners.

The publication of this information does not imply that any licence is granted under any patent or other rights owned by CSR plc.

CSR reserves the right to make technical changes to its products as part of its development programme.

While every care has been taken to ensure the accuracy of the contents of this document, CSR cannot accept responsibility for any errors.

No statements or representations in this document are to be construed as advertising, marketing, or offering for sale in the United States imported covered products subject to the Cease and Desist Order issued by the U.S. International Trade Commission in its Investigation No. 337-TA-602. Such products include SiRFstarIII™ chips that operate with SiRF software that supports SiRFInstantFix™, and/or SiRFLoc® servers, or contains SyncFreeNav functionality.

Life Support Policy and Use in Safety-critical Compliance

CSR's products are not authorised for use in life-support or safety-critical applications. Use in such applications is done at the sole discretion of the customer. CSR will not warrant the use of its devices in such applications.

Performance and Conformance

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