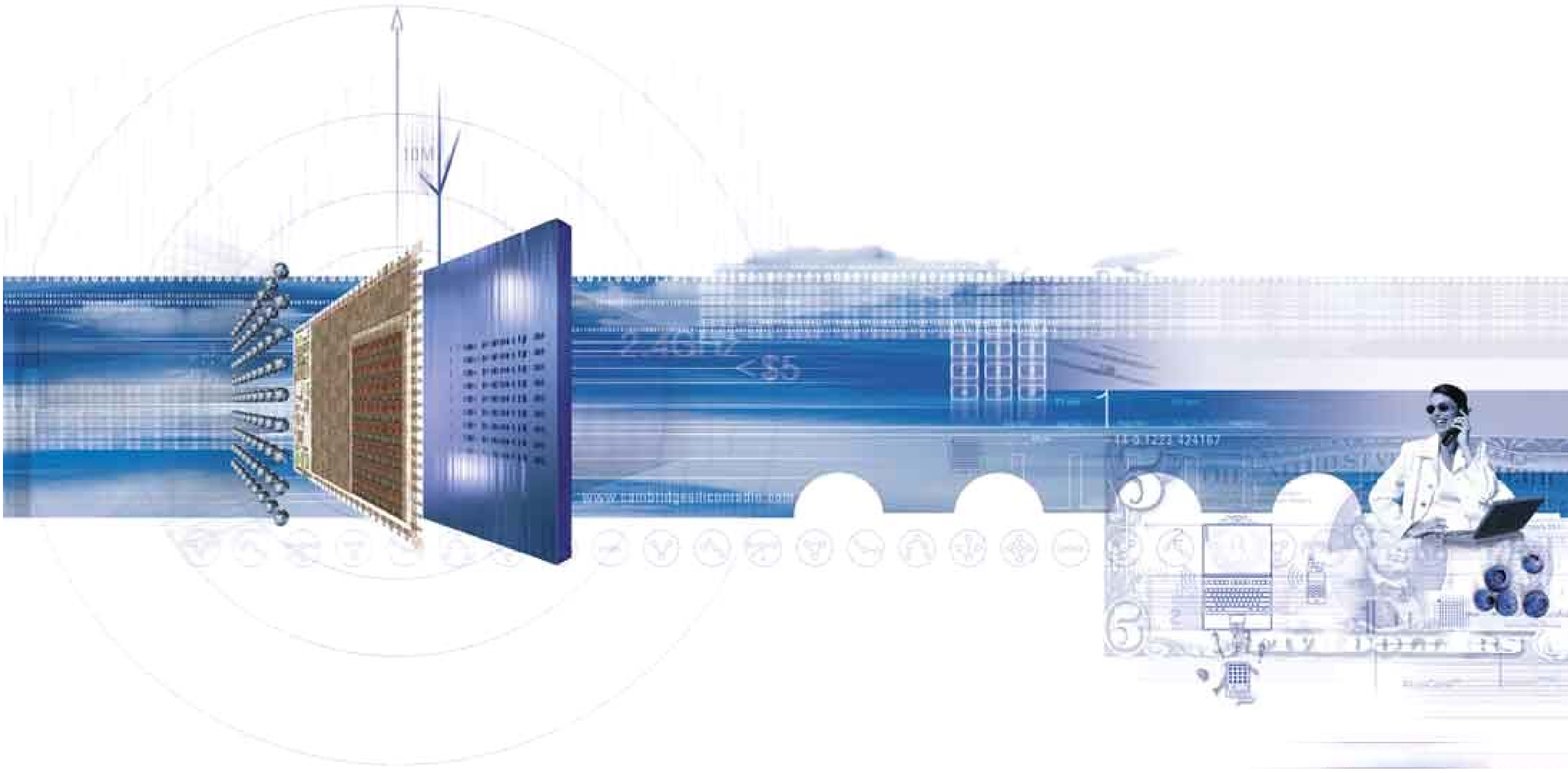
BlueCore™

UART Host Transport Summary

February 2004



CSR

Cambridge Science Park
Milton Road
Cambridge
CB4 0WH
United Kingdom

Registered in England 3665875

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

Contents

List of Figures

List of Tables

BlueCore™ UART Host Transport Summary

1 Introduction

The Bluetooth® specification v1.2 details a number of communication layers and protocols that allow the transmission and reception of data and the control of Bluetooth wireless technology links by the host application. This summary describes the different host transports that can work over the universal asynchronous receiver and transmitter (UART). These include the standard Bluetooth protocols H3 and H4 and the **BlueCore™** Serial Protocol (BCSP) developed by CSR. The summary presents three versions of BCSP designed for embedded systems.

2 Host Transport Overview

The Bluetooth Specification details a protocol for communication between the Host and the Bluetooth hardware called the host controller interface (HCI). Figure 2.1 shows how the protocol layers are commonly split with some layers:

- L2CAP and above residing on the host
- The link manager (LM) and below residing on the Bluetooth hardware

Note:

The Bluetooth hardware is also called the host controller (HC).

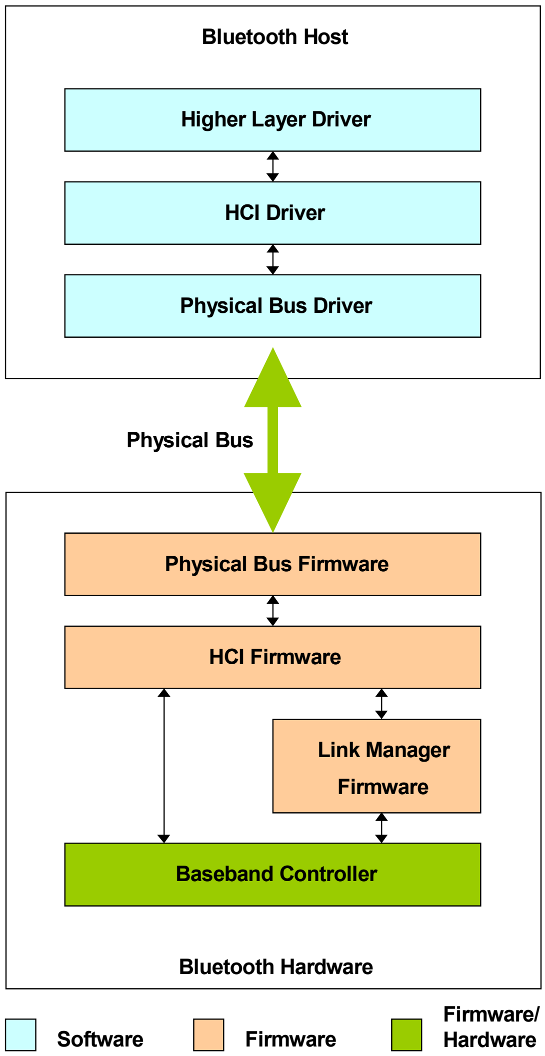


Figure 2.1: Host and Bluetooth Hardware Communications Protocol

HCI defines the control and the bulk of data structures to be passed between the host and the Bluetooth hardware, but it says nothing about how these should be carried between the two units, i.e., HCI is itself transport independent.

The Bluetooth specification defines two host transports to be used to carry HCI data between the host and the Bluetooth hardware over a UART link. These two transports are H3 and H4; in addition, CSR has designed another UART host transport called BCSP as a robust alternative to H4. This section examines all these UART host transports.

2.1 H3 (RS232)

The H3 host transport requires a minimum of three communication lines:

- Receive (Rx)
- Transmit (Tx)
- Common

There are also two optional lines normally used for hardware flow control:

- Ready to Send (RTS)
- Clear to Send (CTS)

Before transmission of HCI data a negotiation packet configures the link for baud rate, parity, stop bit, error response delay (T_{detect}) and protocol.

The Bluetooth specification includes two different protocols:

- Hardware Sync. where the RTS/CTS lines are used for signalling error detection and re-synchronising
- Consistent overhead byte stuffing (COBS) where the hardware flow control lines are optional and error detection and handling is performed via dedicated sync and error messages

When implementing H3, Bluetooth devices can support only one of these options, but hosts must be able to support both. Both protocols involve:

- The receiving end checking incoming messages for the following errors:
  - Parity errors for Hardware Sync.
  - Parity errors and/or cyclic redundancy check (CRC) errors for COBS
- Signal the sending end if they are detected
- Under COBS protocol, only the offending packet is retransmitted
- Under Hardware Sync. transmission is restarted with the offending packet and the original sequence resumed with all subsequent packets being retransmitted in order
- The sending end to maintain a buffer of all transmitted packets until it can be certain that a retransmission will not be required. Certainty can only be achieved when the maximum time for the receiving end to check the packet and signal an error has passed. This is defined in the negotiation packet by T_{detect}.

With its sequencing and error messages and need to retransmit single packets out of sequence, COBS adds an additional processing overhead.

If H3 support is required on a device where a large transmission buffer is available, for example in the case of a personal computer (PC), or the receiving end has a short T_{detect}, then implementation is straightforward. If however, the receiving device has a long T_{detect} or random access memory (RAM) for the buffer is at a premium, then implementation can be very unattractive.

As Bluetooth devices are designed to be low cost, it is hard to justify the extra RAM required for H3, especially given its other disadvantages:

- The extra processing required for COBS resulting in the bandwidth being processor limited to below the maximum required for full speed Bluetooth communications
- The error detection for Hardware Sync is restricted to parity bits

2.2 H4 (UART)

H4 is the simplest of the Bluetooth standard host transports. It is intended to operate over an RS232 link with no parity. Hardware flow control is required. HCI commands are transmitted directly with the addition of a packet indicator to indicate:

- Command Event
- Synchronous connection-oriented (SCO) Data
- Asynchronous connectionless (ACL) Data

The disadvantage with H4 is its poor error detection; with no parity, it can only detect:

- An incorrect HCI packet indicator
- A corrupted HCI command
- The length of an HCI packet is out of range

With the absence of any error recovery strategy when something does go wrong, the only way to recover is to reset the bus and restart communications. This often means reforming the Bluetooth links.

Its advantage is that it is simple, cheap and easy to implement at both the Host and Bluetooth device but it is not robust.

2.3 BCSP

BCSP was developed by CSR to provide a more reliable alternative to the H4 host transport protocol. It operates over a similar RS232 link, but hardware flow control is optional so the number of connections can be reduced from five to three if required. HCI messages are placed in a packet whose format allows the addition of the following features:

- Option for unreliable or reliable links
- For unreliable links there is software flow control and sequencing
- For Reliable links there is acknowledgement of receipt for each packet
- Option for CRC on all messages
- 8-bit checksum on packet headers
- Simple error recovery strategy

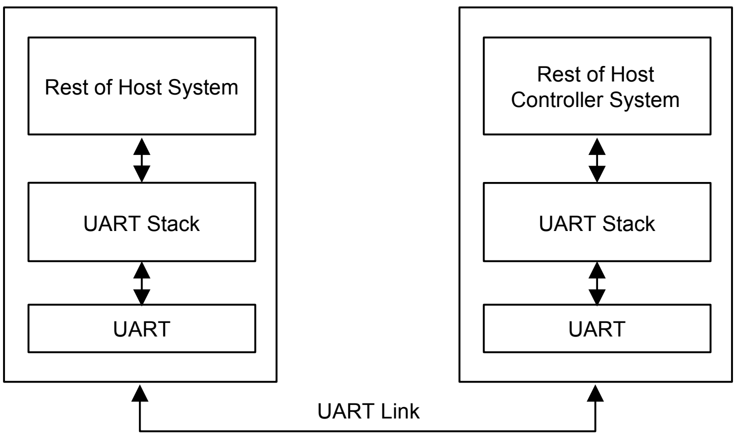


Figure 2.2: UART Stacks

Before any transmission begins, a link establishment procedure permits synchronisation without hardware flow control lines. BCSP packets also include logical channel identifiers to separate Command, Event, ACL and SCO messages. Additional channel allocations also allow direct communications into the upper layers of an on-chip RFCOMM build along with manufacturer specific private channel commands.

An instance of the BCSP stack runs on both the host and the host controller. The top of the BCSP presents:

- One bi-directional reliable datagram service
- One bi-directional unreliable datagram service

Higher layers of the stack can be built upon the two-datagram services.

BlueCore devices contain dedicated hardware to improve performance when using BCSP. The extra information in the BCSP packet header makes this possible.

The hardware checks the packet for completeness and validity, including checksums and CRCs, and routes the payload to its destination using direct memory access (DMA). This reduces the load on the processor to one interrupt per packet with one of two potential benefits:

- More processing power for other tasks
- Greater opportunity for the processor to sleep and reduce current

Whether either of these benefits can be realised will depend on how the BlueCore device is used within the application.

2.4 Comparison Table

Table 2.1 compares the various UART based host transports protocols.

	H3-RS232		H4-UART	BCSP
	COBS	Hardware		
Physical Interface	Serial	Serial	Serial	Serial
Serial Lines Required	3 or 5	5	5	3 or 5
Top Speed (Mbits/s)	0.5	1.0	1.1	1.0
Parity error Check	✓	✓	-	✓
CRC Error Check or Better	✓	-	-	✓
Recovery on Error Detection	✓(1)	✓(1)	No (Reset)	✓
Tolerance of Lost Characters	Good(1)	Average(1)	No	Good
Tolerance of Line Noise	Good	Good	Poor	Good
Works with Hardware Flow Control	✓	-	✓	✓
Works without Hardware Flow Control	✓	✓	-	✓
Support Host Wake-up	-	-	-	✓
Native Support for Other Layers	-	-	-	✓

Table 2.1: Host Transport Layer Comparison

Note:

(1) May require large buffer

3 BCSP Versions

In small embedded applications, BCSP consumes too much RAM. CSR provides other BSSP implementations for different applications.

Currently four versions of BCSP are available:

- Full version, described in the BCSP User Guide
- ABCSP, designed to fit embedded applications and described in ABCSP Overview
- YABCSP, designed to fit embedded applications and described in YABCSP Overview. (YABCSP was written with almost the same interface as ABCSP and shares many of its functions.)
- μBCSP (MicroBCSP), designed to fit embedded applications and described in the μBCSP User Guide

ABCSP, YABCSP and μBCSP are contained in separate zip files, available on the CSR support website at www.csrsupport.com. YABCSP and μBCSP source files include examples of implementations for Microsoft Windows® systems

This section provides a summary of the characteristics and differences between ABCSP, YABCSP and μBCSP. Table 3.1 provides a comparison between these versions of BCSP with H4.

3.1 ABCSP and YABCSP

ABCSP is intended for embedded applications where RAM usage is important. It requires complex integration with its host environment and is biased to minimise its consumption of the host's resources. It complies with the BCSP specification with no restrictions; any violations of the specification should not impact any design implementation.

YABCSP is intended for use in applications that require minimum processing power. The impact of this optimisation is that it uses more and larger chunks of RAM than ABCSP. Hence, in spite of using the same memory management interface, the internal use works differently. It works on internal data buffer (which size is determined by the maximum length of the payload field of a transmitted BCSP packet).

3.1.1 ABCSP and YABCSP Implementation

Code for both ABCSP and YABCSP is divided into two sections, the source code and the interface files. These interfaces are as follows:

- Code common types
- Event reporting
- Unrecoverable errors
- Memory management
- Environment's Transmit message support
- Environment's Receive message support
- Environment's for the Timer functions

ABCSP and YABCSP do not have internal message structure and only deal with message references; therefore, a specific interface is required for message support. As all message management is delegated to the higher layers, including the message storage, a memory management interface is included. For instance, the external code is responsible for allocating the size of the buffer to the ABCSP and YABCSP engines' requirements. The interface consists of functions to:

- Access the raw bytes in a message
- Obtain storage to write bytes into a message
- Obtain buffer to output the UART

Implementing these interfaces consist in replacing macros of the source header files by external environment functions. The role of these macros is well documented in the configuration header files.

A complete description of the interfaces is available in the ABCSP Overview and the YABCSP Overview.

3.1.2 Scheduling ABCSP and YABCSP Operations

The library contains no internal scheduler; it depends on the function calls to drive the code. For example, the transmit path is driven "down" by calls to `abcsp_sendmsg()` and `abcsp_pumptxmsgs()`; these result in calls to `ABCSP_UART_SENDBYTES()`.

Similarly, the receive path is driven "up" by calls to `abcsp_uart_deliverbytes()`, resulting in calls to `ABCSP_UART_SENDBYTES()`.

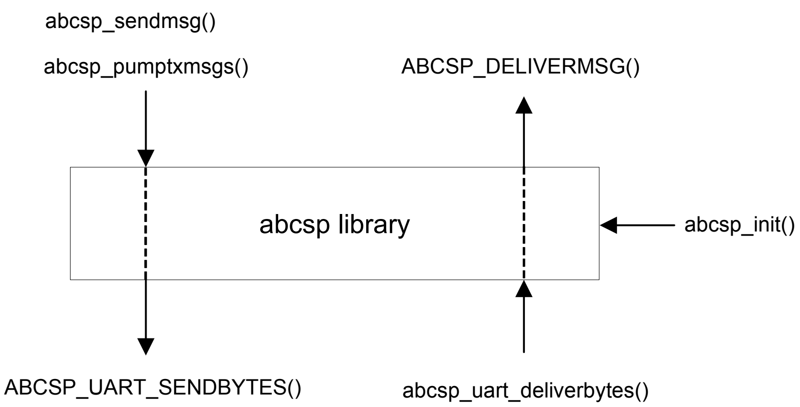


Figure 3.1: ABCSP Library Flow

3.1.3 Configuring ABCSP and YABCSP

- The periods of all the timers can be set separately. These periods are:
  - Link Establishment T_{shy} timer
  - Link Establishment T_{conf} timer
  - BCSP Acknowledgement Timeout timer
- The CRC field is optional on BCSP messages
- The maximum length of the payload field of a received BCSP packet
- The number of BCSP messages that can be handled by the ABCSP library's transmit path at a time

3.2 µBCSP Engine

µBCSP engine is intended for very small embedded environments where the cost of providing program memory space and RAM space are the major primary concerns regarding the impact on overall system costs. µBCSP has a very simple interface and is easy to port. Link establishment is fully implemented and allows the peer reset detection and reporting.

Although µBCSP complies with the BCSP specification, it has one main restriction: the size of the sliding window is set to one packet. This use of the single window reduces the buffering requirements of BCSP and helps to simplify the complexity of the acknowledgement (ACK) scheme. The overall effect of the single window scheme is to reduce the RAM and read only memory (ROM) requirements on the host. The disadvantage of the single sliding window is on the speed of data throughput.

Note:

The µBCSP engine has no timeout and retry mechanism, as this will be system specific.

3.2.1 Implementating µBCSP

The µBCSP interface is composed of the engine interface and the UART interface.

The µBCSP engine interface is based on four functions:

- `ubcsp_initialize()`
- `ubcsp_poll()`
- `ubcsp_send_packet()`
- `ubcsp_receive_packet()`

The UART interface is based on two functions:

- `get_uart()`
- `put_uart()`

µBCSP does not have any interface for message support because it does have an internal message structure. The external message structure has to fit to this internal structure. Consequently, the storage of the message payload is internal to the engine and is based on two separate buffers allocated statically. The external code has no means to control the memory management.

3.2.2 Scheduling µBCSP Operations

µBCSP is based on a polling strategy.

The function `ubcsp_poll()` needs to be processed periodically and it returns as a parameter the state of the engine. This state is represented by a variable, which can take the following values:

- `UBCSP_PACKET_SENT` means a new packet can be sent.
- `UBCSP_PACKET_RECEIVED` means a new packet was received. Another packet will not be received until the ubcsp_receive_packet() function is called again.
- `UBCSP_PEER_RESET` means the BlueCore device has reset.

This function returns a value either showing that an immediate running of the engine is necessary or that a delay before the next process is acceptable.

The period to run `the ubcsp_poll()` has to be carefully chosen. Several parameters have to be taken into account such as baud rate, latency, presence of a FIFO, etc.

3.2.3 Configuring µBCSP

The present version of the µBCSP engine has the following configurations:

- The delay between the return of the polling function and the call to the appropriate µBCSP function.
- The CRC field is optional on BCSP messages.

3.2.4 Code Size

The µBCSP Engine consists of one source file and two header files representing 600 lines of C code. This code compiles and links to:

- 1737 bytes for Pentium PC with CRC
- 1351 bytes for Pentium PC with no CRC

BlueCore™ UART Host Transport Summary

	H4-UART	BCSP	µBCSP	ABCSP	YABCSP
Physical Interface	Serial	Serial	Serial	Serial	Serial
Serial Lines Required	5	3 or 5	3 or 5	3 or 5	3 or 5
Parity Error Check	No	Even	Even	Even	Even
CRC Error Check or Better	No	Yes(1)	Yes(1)	Yes(1)	Yes(1)
Data Lost Detection	No	Yes	Yes	Yes	Yes
Recovery on Error Detection	No(2)	Yes	No(3)	Yes	Yes
Works with Hardware Flow Control	Yes	Yes	Yes	Yes	Yes
Works without Hardware Flow Control	No	Yes	Yes	Yes	Yes
Supports Host Wake-up	No	Yes	Yes	Yes	Yes
Initialise Function Call	`Init()`	`void initialiseStack BCSPStack * stack)`	`ubcsp_initialize()`	`abcsp_init()`	`abcsp_init()`
UART API	`get_uart()`	`ReadFile()`	`get_uart()`	`ABCSP_UART_SENDBYTES (…)`	`ABCSP_UART_SENDBYTES (…)`
	`put_uart()`	`WriteFile()`	`put_uart()`	`abcsp_uart_ deliverbytes (…)`	`abcsp_uart_ deliverbytes(…)`

	H4-UART	BCSP	µBCSP	ABCSP	YABCSP

Stack Organisation, Scheduler	Depends on the implementation.	The stack consists of a core generic engine, which has four I/O points; two byte-oriented buffers and two transfer-request queues. Internally, the stack contains a number of co-operative tasks which share a single stack space and which are managed by a simple scheduler.
		`void (*enterCS)(void *envState)`
		`void (*leaveCS)(void *envState)`
		These two functions are used to enter and leave a critical section. This is required to safely manage the transfer-request queues.
		`void (*signal)(void * envState) ;`
		The signal function is called whenever a new transfer-request is added to a queue by the user. Typically this function is used to wake-up the stack-thread.
			Based on a polling system, the function to call regularly is `ubcsp_poll()`.
			The function `ubcsp_poll()` will return a parameter that indicates whether or not BCSP needs to do any processing. It modifies a variable to signal to the calling function whether a BCSP packet can be sent or received. It is also the function that actually sends or receives a character from the UART.
				The environment is required to provide a set of timers to support for BCSP's retransmission mechanism and for the BCSP link-establishment's timers.
				ABCSP stack requires that ABCSP code should be able to request external code to call internal functions (such as `abcsp_pumptxmsgs()`). Scheduling operations is the responsibility of the developer.
					The environment is required to provide a set of timers to support for BCSP's retransmission mechanism and for the BCSP link-establishment's timers.
					ABCSP stack requires that ABCSP code should be able to request external code to call internal functions (such as `abcsp_pumptxmsgs()`). Scheduling operations is the responsibility of the developer.

	H4-UART	BCSP	µBCSP	ABCSP	YABCSP

Transmit Functions	`SendACLData (…)`
	`SendCommand (…)`
	`SendSCOData (…)`
		`uint16 numFreeSlotsInReceiveBuffer(BCSPStack * stack)`
		`void writeByteToReceiveBuffer(BCSPStack * stack,uint8 data)`
			`ubcsp_send_packet (…)`
			µBCSP uses a window size of one packet therefore only one packet can be sent at a time. The overall effect of the single window scheme is to reduce the RAM and ROM requirements on the host..
				To send a message, higher layer code calls `abcsp_sendmsg()`; this places the message into a queue within the library. The higher layer code then repeatedly calls `abcsp_pumptxmsgs()` to translate the message into its BCSP wire format and push these bytes out of the bottom of the library via `ABCSP_UART_SENDBYTES()`.	To send a message, higher layer code calls `abcsp_sendmsg()`; this places the message into a queue within the library. The higher layer code then repeatedly calls `abcsp_pumptxmsgs()` to translate the message into its BCSP wire format and push these bytes out of the bottom of the library via `ABCSP_UART_SENDBYTES()`.

Receive Functions	`RecvCommand (…)`
	`RecvACLData (…)`
	`RecvSCOData (…)`
		`uint16 numBytesInTransmitBuffer(BCSPStack * stack);`
		`uint8 readByteFromTransmitBuffer(BCSPStack * stack);`
			If the activity argument indicates a packet was received, then this packet needs to be processed in an external function.
			It will then set-up the engine to receive another packet by calling `ubcsp_receive_packet()`.
				For inbound messages the UART driver code passes BCSP wire format bytes into the library via calls to `abcsp_uart_deliverbytes()`. When the library has all of the bytes to form a complete higher layer message it calls `ABCSP_DELIVERMSG()` to pass this to higher layer code.	For inbound messages the UART driver code passes BCSP wire format bytes into the library via calls to `abcsp_uart_deliverbytes()`. When the library has all of the bytes to form a complete higher layer message it calls `ABCSP_DELIVERMSG()` to pass this to higher layer code.

	H4-UART	BCSP	µBCSP	ABCSP	YABCSP
Recovery Mechanism	No	Based on retransmit strategy. Stack internal scheduler is in charge of the retransmission, all the timers functions are implemented. Therefore, no timers interface.	Possible, but not implemented. There is no retransmission mechanism implemented therefore there are no timers.	Based on retransmit strategy. (The environment is required to provide a timer for BCSP's retransmission mechanism). ABCSP reliable messages are placed in a queue which can grow to a fixed maximum length that matches the BCSP transmit window size.	Based on retransmit strategy. (The environment is required to provide a timer for BCSP's retransmission mechanism). ABCSP reliable messages are placed in a queue which can grow to a fixed maximum length that matches the BCSP transmit window size.

Memory Management	Depends on the implementation	`void * (*allocMem)(void * envState,uint32 size);`
		`void (*freeMem)(void * envState,void*);`
		These two functions are used to allocate and free memory.
			Storage of the message payload is internal to the engine (based on two separate buffers allocated statically one for transmit and one for receive) which reduces the code size.	Delegate all the memory management to external code; it asks for access to external buffers: via `ABCSP_xxMSG_GETBUF()`.
				Set of functions for this purpose needs to be written by the developer.
				External code chooses the size of the returned buffer (if a buffer is available!).
					Delegate all the memory management to external code; it asks for access to external buffers: via `ABCSP_xxMSG_GETBUF()`.
					Set of functions for this purpose needs to be written by the developer.

Debug Messages	Depends on the implementation	Several types of macros are used for debugging.	Link Establishment and Error messages	Report significant events to the external environment, two categories informative and panic events.	Report significant events to the external environment, two categories informative and panic events.

	H4-UART	BCSP	µBCSP	ABCSP	YABCSP
Configuration	Depends on the implementation.	Packet acknowledgement timeout. Transmit window size (in packets). (Minimum value is 1. Maximum value is the range of the ACK Field, minus one.) Number of times a message is resent before declaring the link has failed. CRC field is optional on BCSP messages.	The delay between the return of the polling function and the call to the appropriate µBCSP function. The CRC field is optional on BCSP messages.	The periods of all the timers can be set separately (Link Establishment T_{shy} timer, Link Establishment T_{conf} timer, and BCSP Acknowledgment Timeout timer). CRC field is optional on BCSP messages. Maximum length of the payload field of a received BCSP packet. Number of BCSP messages that can be handled by the ABCSP library's transmit path at a time.	The periods of all the timers can be set separately (Link Establishment T_{shy} timer, Link Establishment T_{conf} timer, and BCSP Acknowledgment Timeout timer). The CRC field is optional on BCSP messages. Maximum length of payload field of a received BCSP packet and size of internal buffer for UART output. Number of BCSP messages that can be handled by the ABCSP library's transmit path at a time.
Code Size	Depends on the implementation.	Designed to run on a host system with at least 600 bytes of RAM and approximately 58kb to 8kb of ROM.	Composed of two files (one source and one header). Representing about 600 lines of C code. Compiled under Pentium it represents 1700 bytes with CRC and without CRC about 1300 bytes.	Composed of 36 files into three directories. Representing about 1700 lines of C code. Compiled under Pentium it represents about 2500 bytes with CRC.	Compiled under ARM ADS 1.1 it represents about 7000 bytes with CRC.

Table 3.1: Comparison Between BCSP Variants and H4

Note:

(1) Optional

(2) Reset necessary

(3) Unless recovery strategy is implemented

4 Document References

Document	Reference
Specification of the Bluetooth System	V1.1, 22 February 2001 and v1.2, 05 November 2003
BlueCore Serial Protocol (BCSP)	bcore-sp-012P
BCSP Channel Allocations	bcore-sp-007P
BCSP Link Establishment Protocol	bcore-sp-008P
ABCSP Overview	CSR document AN111
YABCSP Overview	CSR document bcore-an-012P
µBCSP User Guide	CSR document bcor-ug-001P

Acronyms and Definitions

ABCSP	Another BlueCore Serial Protocol
ACK	ACKnowledge
ACL	Asynchronous Connection-Less
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
Bluetooth SIG	Bluetooth Special Interest Group
COBS	Consistent Overhead Byte Stuffing
CRC	Cyclic Redundancy Check
CSR	Cambridge Silicon Radio
CTS	Clear to Send
DMA	Direct Memory Access
FIFO	First In, First Out
HC	Host Controller
HCI	Host Controller Interface
L2CAP	Logical Link Control and Adaptation Protocol (protocol layer)
µBCSP	“Micro” BlueCore Serial Protocol
PC	Personal Computer
RAM	Random Access Memory
RFCOMM	RF Communications
ROM	Read Only Memory
RS232	Serial communications standard
RTS	Ready To Send
Rx	Receive/Receiver
SCO	Synchronous Connection-Oriented
Tx	Transmit/Transmitter
UART	Universal Asynchronous Receiver Transmitter

Record of Changes

Date:	Revision	Reason for Change:
01 MAY 02	a	Original publication of this document (CSR reference: bcore-rp-002Pa)
09 FEB 04	b	Updated to include YABCSP (CSR reference: bcore-rp-002Pb)

UART Host Transport Summary

bcore-rp-002Pb

February 2004