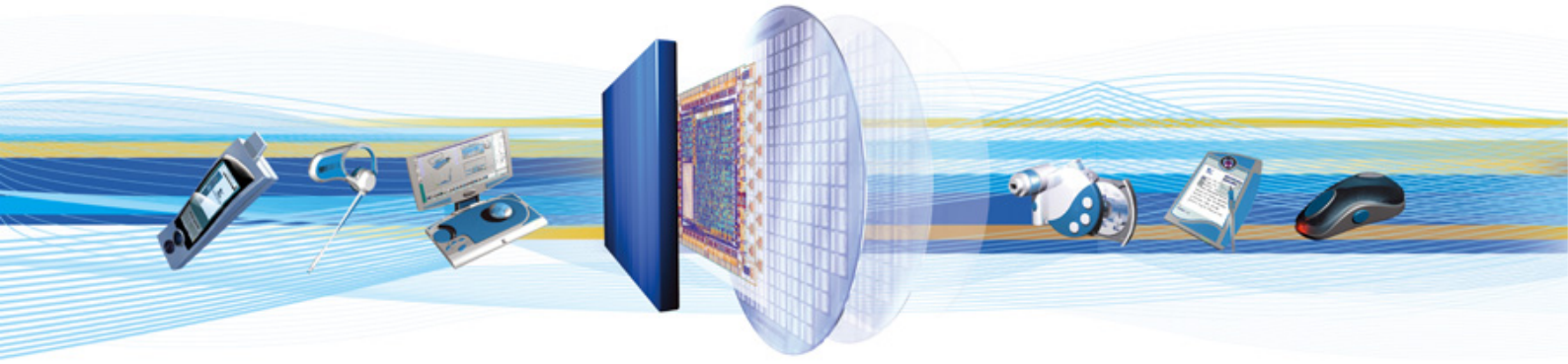
CSR Synergy Bluetooth 18.2.0

Sub-Band Codec

API Description

November 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

1 Introduction

1.1 Introduction and Scope

This document describes the function interface provided by the CSR Synergy Bluetooth Sub-Band Codec (SBC) implementation. The Bluetooth SBC codec is described in the Advanced Audio Distribution Profile, ref. [A2DSPEC] appendix B.

1.2 Assumptions

In the following document, it is assumed that audio samples are expected to be 16 bit wide. Furthermore, it is assumed that the user has knowledge of the Advanced Audio Distribution Profile specification [A2DSPEC].

2 Description

2.1 Introduction

SBC is a low computational complexity audio codec specially designed for Bluetooth audio applications enabling a moderate bit rate. SBC uses 4 or 8 sub-bands, a cosine-modulated filter bank for analysis and synthesis, an adaptive bit allocation algorithm and simple adaptive block PCM quantizers.

The SBC is highly configurable which allows the audio quality and bit rate to be adjusted to match the available Bluetooth bandwidth. Initially, this is done by selection of number of sub-bands (4 or 8), number of blocks (4, 8, 12 or 16), bit allocation algorithm (SNR or loudness), channel mode (mono, dual channel, stereo or joint stereo) and sample frequency (16, 32, 44.1 or 48 kHz). When an audio stream is established these parameters must be configured and remains fixed for the duration of the stream unless the stream is temporarily suspended and reconfigured. While streaming, the bit rate may be adjusted by the bit pool, the number of bits per SBC frame used for carrying encoded audio data.

2.2 Overview

The SBC implementation which is part of the advanced audio demo application of CSR Synergy Bluetooth can be seen as a library which the application can use for the decoding and encoding of SBC streams. The SBC implementation is suited to run efficiently on any general-purpose processor.

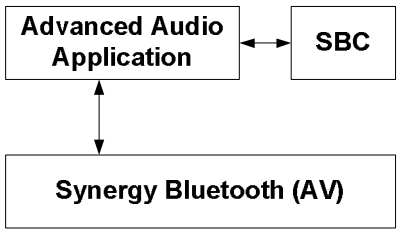


Figure 1: SBC reference model

The SBC API operates on SBC stream handles. An SBC stream handle must be obtained in order to use the API for encoding or decoding streams. The following functions are used for obtaining and discard stream handles:

- CsrBtSbcOpen
- CsrBtSbcClose

The following functions are used for encoding:

- CsrBtSbcConfig
- CsrBtSbcCalcBitPool
- CsrBtSbcEncode

The following functions provide functionality for decoding:

- CsrBtSbcInitDecoder
- CsrBtSbcReadHeader
- CsrBtSbcDecode

Furthermore, a number of functions are available for accessing SBC stream configuration parameters:

- CsrBtSbcGetChannelMode
- CsrBtSbcGetAllocMethod
- CsrBtSbcGetSampleFreq
- CsrBtSbcGetNumBlocks
- CsrBtSbcGetNumSubBands

- CsrBtSbcGetBitPool

The following section describes the interface functions between the advanced audio application and the SBC codec implementation.

3 Interface Description

3.1 CsrBtSbcOpen

This function is used for obtaining an SBC stream handle.

*void *CsrBtSbcOpen()*

The function does not accept any parameters. The return value is a pointer to the SBC stream handle. The stream handle is *not* initialised with valid settings, and when no longer needed it should be discarded using CsrBtSbcClose to avoid leaking memory.

3.2 CsrBtSbcClose

This function is used for discarding SBC stream handles.

void CsrBtSbc*Open(* *void **handlePtr)*

The valid parameters are as follows:

handlePtr: a pointer to a pointer to an SBC stream handle

**handlePtr* should point to an SBC stream handle previously obtained with CsrBtSbcOpen. When returning, **handlePtr* will point to NULL to make debugging easier by ensuring that further use of **handlePtr* will fail.

3.3 CsrBtSbcConfig

The SBC encoder must be configured with all adjustable parameters before audio samples are fed into it. Therefore, when an audio stream is established or when the stream is reconfigured, the *CsrBtSbcConfig* function should be called to set the SBC parameters agreed upon between the audio source and sink.

```
CsrUint16 CsrBtSbcConfig(   void                *handle,
                            CsrBtSbcChannelMode  channel_mode,
                            CsrBtSbcAllocMethod  alloc_method,
                            CsrUint16            sample_freq,
                            CsrUint8             nrof_blocks,
                            CsrUint8             nrof_subbands,
                            CsrUint8             bitpool   )
```

The valid parameters are as follows:

handle:	a pointer to an SBC stream handle
channel_mode:	mono (0), dual channel (1), stereo (2) or joint stereo (3)
alloc_method:	Loudness (0) or SNR (1)
sample_freq:	16000, 32000, 44100 or 48000 Hz
nrof_blocks:	4, 8, 12 or 16 blocks
nrof_subbands:	4 or 8 sub-bands
bitpool:	number of bits in a SBC frame representing the audio

For mono and dual channel streams the bit pool value must be in the range from 2 to *16*nrof_subbands*. For stereo and joint stereo streams the valid range is from 2 to *32*nrof_subbands*.

If the configuration parameters are accepted *CsrBtSbcConfig* returns the SBC frame length based on the given parameters. If the parameters are not accepted, it returns zero.

3.4 CsrBtSbcCalcBitPool

An initial bit pool value can be calculated from a targeted audio bit rate with the function *CsrBtSbcCalcBitPool.* The bit pool value is a more dynamic value than the other SBC configuration parameters as it may be changed

from frame to frame to match the available bandwidth which may vary over time depending on the Bluetooth radio conditions.

```
CsrUint8 CsrBtSbcCalcBitPool(   CsrUint8                *bitPoolAlt,
                                CsrUint8                *togglePeriod,
                                CsrBtSbcChannelMode_t   channel_mode,
                                CsrUint16               sample_freq,
                                CsrUint8                nrof_blocks,
                                CsrUint8                nrof_subbands,
                                CsrUint16               bitrate)
```

The function returns the bit pool value that corresponds best to the specified bit rate and configuration. It also returns an alternative bit pool (bitPoolAlt) and a toggle period (togglePeriod) that may be used for obtaining the specified bit rate more accurately. If used, the alternative bit pool should be used once every toggle period frames instead of the normal bit pool. If the toggle period returned is zero, the alternative bit pool should not be used.

3.5 CsrBtSbcEncode

When the SBC codec has been configured, the *CsrBtSbcEncode* function can be called to encode audio samples into SBC frames.

*CsrUint16 CsrBtSbcEncode(void *handle, const CsrInt16 audio_samples[][2], CsrUint8 *frame, CsrUint8 bit_pool)*

The function is given an SBC stream handle, the audio samples (16 bit samples) in a two-dimensional array (*audio_samples[][2]*) corresponding to the left and right channels being stored in an interlaced manner. The number of samples that must be given for each call to *CsrBtSbcEncode* for encoding one SBC frame can be calculated from the following formula:

*nrof_samples = nrof_blocks * nrof_subbands*

E.g. encoding one stereo SBC frame with 8 sub-bands and 16 blocks requires 128 16-bit samples for each channel (left and right).

The bit pool value (*bit_pool*) specifies the compression level of the audio, i.e. the number of bits in the SBC frame that should be used for representing the sampled audio.

The *CsrBtSbcEncode* function returns a positive value upon success indicating the size of frame created. The generated SBC frame is written to the location pointed to by the parameter **frame*.

3.6 CsrBtSbcInitDecoder

Prior to start of decoding, a call should be made to this function to reset the decoder to its initial state.

void *CsrBtSbcnitDecoder(void *handle);*

3.7 CsrBtSbcReadHeader

This function is used for validating the header of a received SBC frame. It must be called for each SBC frame prior to its decoding as it also picks out the scale factors included in the SBC frame header which must be used by the decoder.

*CsrUint16 CsrBtSbcReadHeader(void *handle, CsrUint8 *frame);*

As input to the function a stream handle and the location of the SBC frame must be specified. If the SBC frame is found to be invalid it returns the value 0 and otherwise the length of the SBC frame.

3.8 CsrBtSbcDecode

The *CsrBtSbcDecode* function decodes a SBC frame into audio samples.

*void CsrBtSbcDecode(void *handle, CsrUint8 *frame, CsrUint16 audio_samples[16][2][8])*

The frame to decode is pointed to by *frame and the resulting decoded audio samples are returned in *audio_samples[blocks][channels][subbands].* Often the audio samples returned will need some restructuring so that the left and right channel samples are interlaced and ordered chronologically. An example of a routine to do this task, is as follows:

```
for (b=0; b<nrof_blocks; b++)
{
   for (sb=0; sb<nrof_subbands; sb++)
   {
      for (ch=0; ch<nrof_channels; ch++)
      {
         *audio_ptr++ = audio_samples[b][ch][sb]; /* placing samples interlaced and chronologically */
         audio_length += 2;                  /* counting the length of samples in bytes          */
      }
   }
}
```

3.9 CsrBtSbcGetChannelMode

This function returns the channel mode the stream is configured to use.

*CsrBtSbcChannelMode CsrBtSbcGetChannelMode(void *handle)*

The valid parameters are as follows:

handle: a pointer to an SBC stream handle

handle should point to an SBC stream handle previously obtained with CsrBtSbcOpen.

3.10 CsrBtSbcGetAllocMethod

This function returns the current allocation mode used with the specified SBC stream.

*CsrBtSbcAllocMethod CsrBtSbcGetAllocMethod(void *handle)*

The valid parameters are as follows:

handle: a pointer to an SBC stream handle

handle should point to an SBC stream handle previously obtained with *CsrBtSbc*Open.

3.11 CsrBtSbcGetSampleFreq

This function returns the current sample rate used with the SBC stream.

*CsrUint16 CsrBtSbcGetSampleFreq(void *handle)*

The valid parameters are as follows:

handle: a pointer to an SBC stream handle

handle should point to an SBC stream handle previously obtained with *CsrBtSbc*Open.

3.12 CsrBtSbcGetNumBlocks

This function returns the number of blocks used with the SBC stream.

*CsrUint8 CsrBtSbcGetNumBlocks(void *handle)*

The valid parameters are as follows:

handle: a pointer to an SBC stream handle

handle should point to an SBC stream handle previously obtained with CsrBtSbcOpen.

3.13 CsrBtSbcGetNumSubBands

This function returns the number of subbands the stream is configured to use.

*CsrUint8 CsrBtSbcGetNumSubBands(void *handle)*

The valid parameters are as follows:

handle: a pointer to an SBC stream handle

handle should point to an SBC stream handle previously obtained with CsrBtSbcOpen.

3.14 CsrBtSbcGetBitPool

This function is used for discarding SBC stream handles.

*CsrUint8 CsrBtSbcGetBitPool(void *handle)*

The valid parameters are as follows:

handle: a pointer to an SBC stream handle

handle should point to an SBC stream handle previously obtained with CsrBtSbcOpen.

4 Document References

Document	Reference
Advanced Audio Distribution (A2D) Profile Version: 1.0 Date: 22-05-2003	[A2DSPEC]

Terms and Definitions

AV	CSR Synergy Bluetooth Audio/Video profile component
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
SBC	Sub-Band Codec
SNR	Signal-to-Noise Ratio
UniFi™	Group term for CSR's range of chips designed to meet IEEE 802.11 standards

Document History

Revision	Date	History
1	26 SEP 11	Ready for release 18.2.0

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.

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.