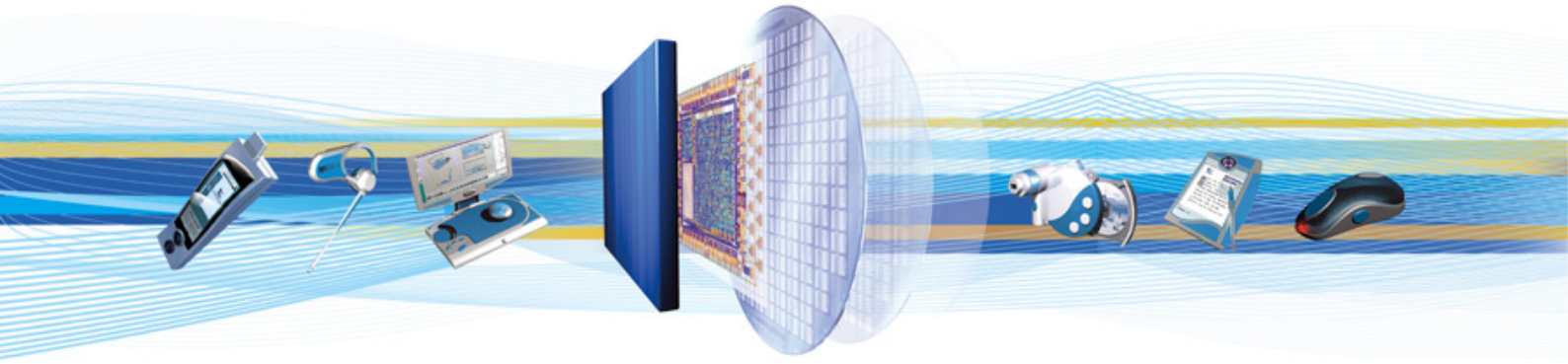
CSR Synergy Bluetooth 18.2.0

AV Implementation Guide

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

List of Tables

1 Introduction

1.1 Document Objective

The purpose of this document is to provide guidance on how to implement the AV profile application layer using the CSR Synergy Bluetooth A2DP profile layer. The document contains hints and notes to take into consideration during the design and implementation process.

1.2 Overview

The objective of the A2DP profile is to stream high-quality audio from a source to a sink in real-time. In a typical use case, the audio source is a mobile handset (mobile phone, PDA etc.) and the audio sink is a stereo headset or speakers. An audio stream is a logical end-to-end connection through which audio content data flows at an average bit rate corresponding to the audio bit rate. The audio is encoded by the source application prior to being sent on the stream and decoded at the arrival in the sink application in order to decrease the bandwidth requirements between the devices. Concurrent with the A2DP stream, the AV remote control profile (AVRCP) may exist in order to send commands between the sink and source devices that can control e.g. start and stop of the data flow on the stream, changing audio tracks etc. A sink may indirectly control the source using the AVRCP profile by sending basic commands to the source. The A2DP and AVRCP profiles are to be understood as independent; the inter-action between the two profiles is made at application level.

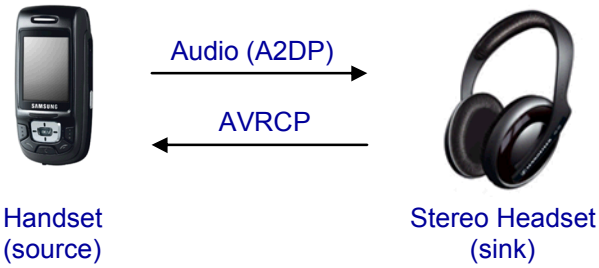


Figure 1: Typical handset to headset setup

Both the source and the sink are required to be able to establish a stream. A2DP defines two roles besides the source and sink roles; an initiator and an acceptor role. The initiator and acceptor roles are independent of the source and sink roles. The device that initiates a procedure is considered the initiator and the other the device the acceptor. The initiator and acceptor roles may change during the stream lifetime.

To manage the stream a set of procedures is defined in the specification that allows the devices to explore each other's stream capabilities, establish and remove streams. The set of procedures and their purpose in the life cycle of a stream is illustrated in Figure 2 and will be described briefly in the following. Each description will also describe how the particular state or event is handled by the A2DP demo supplied with CSR Synergy Bluetooth.

The *discover* procedure is used for retrieving information from the peer device about its stream end-points. A stream end-point has attributes assigned to it that tells about its type (source or sink), its media type (audio or video) and its availability (in use or not). The acceptor of this procedure returns a list of its end-points and attributes, each assigned a stream end-point identifier. A device will have minimum one end-point for each supported codec.

Following the *discover* procedure is usually one or more *get capabilities* procedures to retrieve the details of those end-points which suits the needs of the initiator. Each *get capabilities* procedure retrieves the details of one end-point identified by its identifier returned in the *discover* procedure. The acceptor returns the details, i.e. codec type (e.g. SBC, MPEG, AAC etc.) and the codec specific parameters. The initiator will retrieve the details for all the end-points of interest or until a suitable end-point is found. If a device already knows the end-point details of the remote device it need not perform the *discover* and *get capabilities* procedures prior to establishing a stream. In the A2DP demo, the codec filter will either be requested to return it's capabilities (*f_get_caps*), or it will receive the remote endpoint capabilities and be expected to return the desired configuration (*f_remote_caps*).

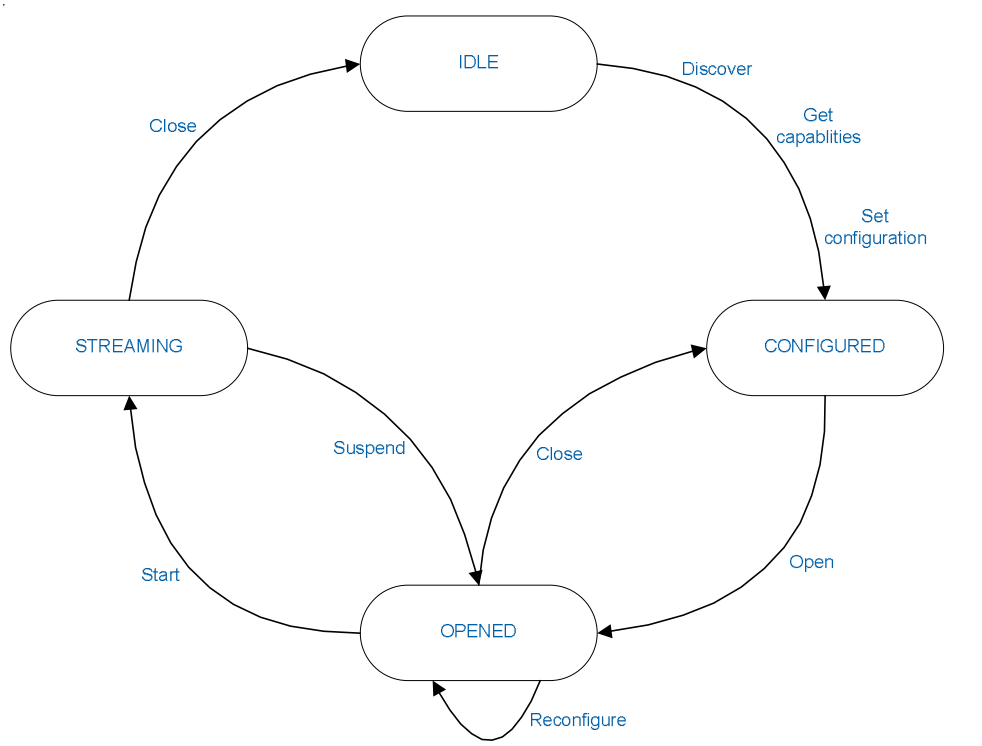


Figure 2: Stream life cycle

Once the end-points are known, the stream establishment can begin by the *set configuration* procedure. The initiator assigns a stream configuration by this procedure to the selected end-point of the remote device and one of its own end-points. The configuration contains information about sampling frequency, upper and lower bandwidth limits and other codec specific parameters. If the acceptor can agree to the configuration given by the initiator the stream enters CONFIGURED state. In the A2DP demo, the remote side can request a specific configuration using the codec function *f_set_config*, and the codec can also be required to return it's current configuration using *f_get_config*.

The next step in the stream establishment is the *open* procedure that will open a separate transport channel for the stream data. The initiator of the *open* procedure must be the same as the device who initiated the *set configuration* procedure. The state will be OPENED afterwards. In the A2DP demo, the *f_open* function will be called.

The last step that completes the stream establishment is the *start* procedure that triggers the start of sending data on the stream. Initially, it is only the device that initiated the *open* procedure that is allowed to initiate the *start* procedure. The state of the stream is now STREAMING, meaning that media content is now allowed to flow from source to sink. The A2DP demo calls the codec function named *f_start* to signal that streaming should commence.

If the audio content changes, e.g. due to changing tracks on a play list, it may be needed to reconfigure the stream settings to match the content. However, this is only allowed when the stream is in OPENED state which is accomplished by the *suspend* procedure when in STREAMING state. The *suspend* procedure is optional to support and thus suspending might fail. The only alternative is then to close the stream and re-establish a new stream with the new configuration. The A2DP demo will always signal the codecs and audio endpoint system that streaming has stopped using the *f_stop* function.

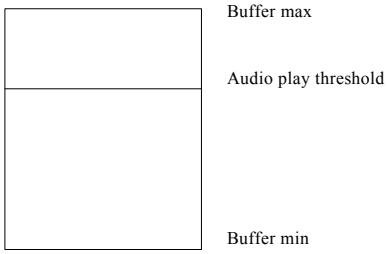
The *reconfigure* procedure is used for changing the stream configuration from the previous configuration. The procedure can only be used for changing codec parameters, - not the codec type since this requires a change of end-points. Just as the *suspend* procedure the *reconfigure* procedure is optional and thus might fail. The initiator of the *suspend* procedure must also be the initiator of the *reconfigure* procedure. The A2DP demo handles reconfiguration using the *f_get_caps*, *f_remote_caps*, *f_get_config*, and *f_set_config* functions. When the new configuration has been agreed upon, the *f_open* function is called. Note that this implies that *f_open* can be called multiple times without corresponding *f_close* calls.

The streaming is resumed by the *start* procedure. Again only the initiator of the *suspend* procedure is allowed to initiate resume. In the demo, this is done using the *f_start* function.

Both devices may close the stream by the *close* procedure at any time, releasing all resources used by the stream. The A2DP closes the stream using *f_close*.

2 Real Time Audio Streaming

Streaming is the real time transfer of media data with a certain bit rate. This is unlike a file transfer, where the data is transmitted as fast as possible and without a specific requirement to the min./max transfer rate.



In a well-behaved audio source implementation, the audio data is streamed evenly spaced over time and is not sent in bursts. Bursting must be avoided as a headset has a limited buffer and bursting may cause buffer overflow and data discarding. Large buffers are often not wanted on the headset in order to lower the HW cost and to minimize the audio delay. The latter is important if for instance the headset is used for real time audio notification like ring tones and system sounds. Typically, the headset has a buffer capable of storing audio for less than 200 msec. Taking into account that the headset also needs some buffer before starting to play the audio samples, this may leave less than 100 msec of buffer space for storing data bursts.

2.1 Timing

Accurate timing is essential to ensure good audio quality. In this context, accurate timing is two fold:

- timing of individual packets
- average data throughput

2.1.1 Packet Timing

This is the timing of each audio packet relative to the previous audio packet sent. If an audio packet contains data for n msec of audio, then the next audio packet must in average be sent n milliseconds later.

If the platform supports periodic high precision clock interrupts, this method can be used for packet timing, i.e. call a transmit function each time an interrupt is generated. If the platform only supports timers that expire once, the time used for invoking the transmit function must be taken into account when restarting the timer:

```
Calculate time for each packet

transmit_av_packet()
{
    Restart timer (time for one packet – invoking time)
    Send data
}
```

If the platform supports only soft timing, the real time clock of the system can be used instead. The following scheme can be used:

```
transmit_task()
{
  Calculate time for each packet

  Repeat
  {
    Send data
    Wait until time > (sent packets + 1) * time for one packet
  }
}
```

The transmit task must be given a sufficiently high priority to ensure that data is sent before the buffer on the headset underflows. Typically, this implies that the sending task must be serviced within a window of 2 data packets (depending on the packet size used).

2.1.2 Data Throughput

This is the average number of packets sent over time. The used timer must have accuracy better than 0.1%. If this accuracy is not complied with, the buffer in the headset may under or overflow.

2.1.3 AV Source Stream Buffer Levels

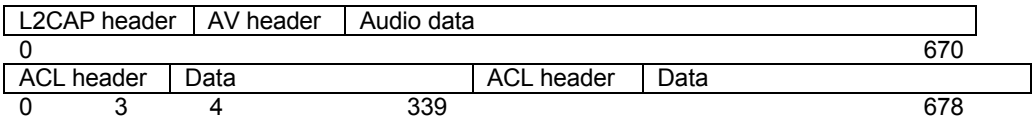
The CSR Synergy Bluetooth AV profile will seek to help the source side by periodically sending AV_QOS_IND messages to the registered AV user task. These messages contain a member named *bufferStatus* that is a number between 0 and 9. The number represents the ring buffer utilization in the AV source streamer and can be used by the user application as a guideline for adjusting the bitrate or configuration of the stream data.

For example, if the buffer status runs continuously at level 0, this indicates that remote sink is able to consume all data. If possible, the SBC bit pool could be increased to allow for better audio quality at the expense of more bandwidth usage. Likewise, if buffer status levels starts to increase, this indicates that the AV streamer is not able to send data to the remote sink fast enough, and the SBC bit pool can be decreased at the expense of lower audio quality but also less bandwidth usage.

A simple scheme for utilising the AV_QOS_IND messages have been implemented in the A2DP demo and SBC codec that dynamically seeks to adjust the bit pool.

2.2 L2CAP Packet Size

Bluetooth defines several packet types to be used on the air link (DH1, DH3 etc). These packets can carry different payload size and take up different air time. The individual packet types take up the same amount of air bandwidth no matter how much data is actually filled into the packets. In order to make efficient use of the air bandwidth it is recommended to fill up the packets as much as possible. This can be achieved by selecting the proper L2CAP packet size and filling up the AV data signal. Normally a L2CAP packet size of 672 bytes is recommended.



The data-fitting algorithm should take care of filling the AV data signal with as much data as possible.

It is not recommended to limit the allowed packet types on the air interface to only DH5 packets; this is handled by the link manager in the BlueCore™ depending on the actual characteristics of the link quality.

3 Audio Encoding

Encoding is used for minimising the needed bandwidth required for transferring the audio data. Thus, the encoding will compress the audio before transmission over the air. The drawback is that compression also implies losing information and therefore a lower audio quality compared with the original audio. However, if an appropriate encoding algorithm known as the codec is selected and the encoded audio bit rate is high enough, this will not be perceptible by the human ear.

On the AV source device, the encoder is responsible for encoding raw audio samples reducing the amount of data necessary to represent the audio and thereby reducing the required bandwidth to stream the audio.

Several codecs can be used for this purpose, e.g.:

- SBC (Bluetooth mandatory)
- MP3
- WMA
- …

The Bluetooth Advanced Audio Distribution profile mandates support for SBC encoding only but other codecs can be used. It is up to the audio source and sink to agree on the used codec upon stream establishment, but SBC must be supported by both and can therefore be used as a fallback in case the source and sink can not agree to use other codecs.

3.1 Sub-Band Codec (SBC)

Audio files are rarely stored in SBC format but often in other popular formats like MP3 and WMA, and will therefore require both decoding of the format it is stored in and encoding to SBC on the source side before streaming to the headset.

Characteristics:

- Requires high processing power due to e.g. MP3 decoding followed by SBC encoding (see Table 2 containing Dhrystone 2.1 measures for the CSR Synergy Bluetooth SBC encoder implementation)
- Quality of encoded audio can be adjusted to match link quality
- Royalty free
- High quality audio requires a bit rate of 300 kbps or higher.
- The required bit rate can be reduced at a cost of sound quality. A bit rate higher than 240 kbps is recommended

When a SBC stream is established, a SBC codec configuration must be chosen, i.e. sampling frequency (16, 32, 44.1 or 48 kHz), channel mode (mono, dual channel, stereo, or joint stereo), number of sub-bands (4 or 8), number of blocks (4, 8, 12 or 16) and bit allocation method (SNR or loudness). The choice of configuration should be based on the parameters supported by both the audio source and sink (headset) and the available audio format, e.g. if audio is only available at a certain sampling frequency or channel mode. The mandatory SBC parameter values that must be supported for source and sink devices are listed in the table below.

SBC parameter	Source	Sink
Sampling frequency	At least one of 44.1 or 48 kHz	44.1 and 48 kHz
Channel mode	Mono and at least one of dual channel, stereo or joint stereo	All (mono, dual channel, stereo, and joint stereo)
Block length	All (4, 8, 12 and 16)	All (4, 8, 12 and 16)
Subbands	8	All (4 and 8)

SBC parameter	Source	Sink
Allocation method	Loudness	All (SNR and loudness)

Table 1: Mandatory SBC parameter value support for source and sink

When the SBC codec has been configured, the codec is ready to encode samples into SBC frames. To encode exactly one SBC frame a total of 'subbands' * 'blocks' samples are required. The SBC encoder expects to be given the samples as 16 bit wide samples with both channels interlaced in memory, see figure below.

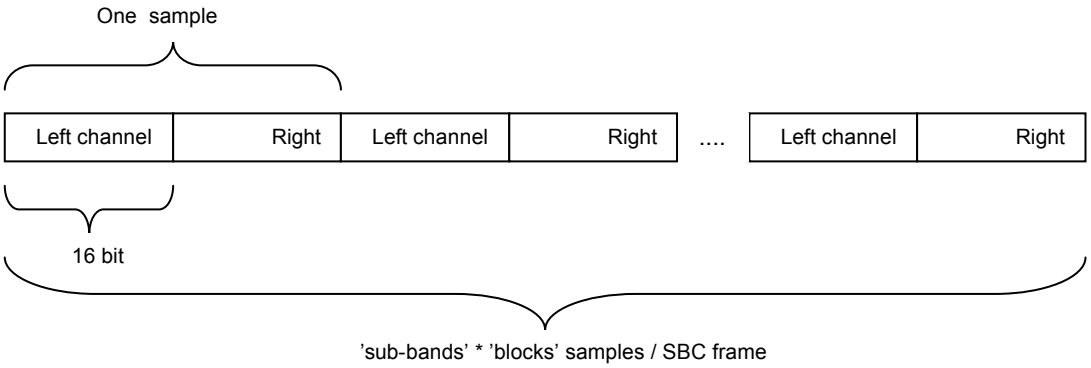


Figure 3: SBC sample structure

The A2DP SOURCE application may have to provide some mechanism to up-sample the source audio stream. The SINK device (typically a stereo headset) only has to support sampling frequencies of 44.1 kHz and 48 kHz. If the source audio stream is sampled with e.g. 22.05 kHz, it is the responsibility of the A2DP application to either reject streaming the audio or to up-sample the audio to a supported sampling frequency.

For each encoding a bit pool parameter is specified to the SBC codec. The bit pool indicates the amount of bits in a SBC frame that can be used for representing the audio signal after encoding/compression. The quality of the encoded audio increases with increasing bit pool value. The bit pool can be adjusted unilaterally from frame to frame by the audio source or may be fixed during the lifetime of the SBC stream. For mono and dual channel streams the bit pool must be in the range from 2 to 16 * 'sub-bands'. For stereo and joint stereo streams the valid range is from 2 to 32 * 'sub-bands'. In addition, the bit pool value must also be within the minimum and maximum values indicated by the headset's end-point capabilities.

The following formulas can be applied for conversion between bit rate and bit pool and calculation of the resulting frame length.

For mono and dual channel streams as:

$$(1)\quad bit\_pool = \left\lfloor \frac{bit\_rate * subbands * blocks / sample\_freq - 4 * subbands * channels - 32}{blocks * channels} \right\rfloor$$

$$(2)\quad bit\_rate = \frac{sample\_freq}{subbands * blocks} * (bit\_pool * blocks * channels + 4 * subbands * channels + 32)$$

$$(3)\quad frame\_len = 4 + \frac{subbands * channels}{2} + \left\lceil \frac{blocks * channels * bit\_pool}{8} \right\rceil$$

and for stereo and joint stereo streams (j=1 for joint stereo, otherwise 0):

$$(4)\quad bit\_pool = \left\lfloor \frac{bit\_rate * subbands * blocks / sample\_freq - 8 * subbands - 32 - j * subbands}{blocks} \right\rfloor$$

$$(5)\quad bit\_rate = \frac{sample\_freq}{subbands * blocks} * (bit\_pool * blocks + 8 * subbands + 32 + j * subbands)$$

$$(6)\quad frame\_len = 4 + subbands + \left\lceil \frac{blocks * bit\_pool}{8} + \frac{j * subbands}{8} \right\rceil$$

EXAMPLE: *As an example, given an SBC stream with the following configuration:'*

Channel mode: *Joint stereo*
Sample frequency: *48 kHz*
Subbands: *8*
Blocks: *16*

If the wanted bit rate should be ~300 kbps, equation 4 is used for finding the resulting bit pool value:

$$bit\_pool = \left\lfloor \frac{300000 * 8 * 16 / 48000 - 8 * 8 - 32 - 1 * 8}{16} \right\rfloor = 43$$

and the SBC frame length is then calculated from equation (6):

$$frame\_len = 4 + 8 + \left\lceil \frac{16 * 43}{8} + \frac{1 * 8}{8} \right\rceil = 99 \text{ bytes}$$

To utilise the bandwidth most efficiently, as many SBC frames should be included in each media packet (L2CAP payload) as possible:

$$(7)\quad frames\_per\_l2cap\_packet = \left\lfloor \frac{L2CAP\_MTU - 13}{frame\_len} \right\rfloor$$

The time between each transmission of an AV media packet is then given by:

$$(8)\quad time\_between\_tx\_sbc = \frac{frames\_per\_l2cap\_packet * subbands * blocks}{sample\_freq}$$

*In the above example, if the L2CAP MTU is e.g. 672 bytes then the number of SBC frames to be included in each AV media stream packet (L2CAP payload) should be six (6). That means that for every 16 milliseconds an AV media packet with 6 SBC frames should be generated and sent from 768 samples (6*subbands*blocks).*

The SBC encoding is a relatively computation heavy task that must be able to run in real-time. The encoding must be able to complete within the duration between each media packet transmission. For reference, the Dhrystone 2.1 MIPS usage of the polyphase analysis implementations that are part of the SBC encoder supplied with CSR Synergy Bluetooth have been measured on an ARM946E-S based platform. The results are listed in Table 2 below. All results are approximate.

SBC encoder (polyphase analysis) implementation	**4 subbands**	**8 subbands**
ANSI C (polyphase_analysis.c)	13.4	13.0
ARM assembler with DSP extensions, loops not unrolled (polyphase_analysis_arm9e_comp.s)	11.7	10.5

SBC encoder (polyphase analysis) implementation	4 subbands	8 subbands
ARM assembler with DSP extensions (polyphase_analysis_arm9e.s)	10.8	9.5

Table 2: Dhrystone MIPS usage for SBC encoder implementations

If possible, it is recommended to execute the polyphase analysis code from a fast RAM area.

In addition to the SBC encoder, also the transmission of the packets through CSR Synergy Bluetooth to the BlueCore chip will consume a significant amount of MIPS. The actual MIPS usage depends highly on the scheduler and UART driver implementation as well as the selected audio bit rate. A typical usage is in the range 7 to 10 Dhrystone 2.1 MIPS.

It is recommended to run the SBC encoding in a separate thread with a lower priority than the core Bluetooth protocol stack. The encoding thread must encode the audio to a temporary output buffer from where the Bluetooth stack can retrieve the already encoded SBC frames and transmit it. The reason for this is to have as little latency for the core communication thread as possible.

3.2 MPEG 1/2 Layer 3 (MP3)

The MPEG1/2 layer 3 codec also known as MP3 codec is optional to support and the use of the MP3 codec therefore depends on whether it is supported by both the source and the sink device. If the sink supports the MP3 codec, no encoding is needed on the source if the audio is stored as MP3 file, which is likely as MP3 is one of the most commonly used audio file formats. An MP3 file may be streamed directly from the file, frame by frame as long as the MP3 file has been encoded with codec parameters supported by the sink.

Characteristics:

- Higher codec compression rate than SBC (less bandwidth requirement for equivalent audio quality). A bit rate of 128 kbps is usually sufficient to obtain high quality audio
- Royalty
- Variable bit rate can be used
- Lower power consumption as processing requirements and bandwidth requirements are lower

If one MP3 frame is transmitted in each media packet, the timing of transmissions can be calculated using the knowledge that each MP3 frame represents exactly 1152 or 576 samples for MPEG 1 and MPEG 2, respectively.

$$(9) \quad time\_between\_tx\_mpeg1 = \frac{1152}{sample\_freq}$$

$$(10) \quad time\_between\_tx\_mpeg2 = \frac{576}{sample\_freq}$$

If one MP3 frame is too large to fit into one L2CAP frame, two (or more) L2CAP frames must be transmitted right after each other. E.g.:

```
frame_size = mp3_frame_size;
while(frame_size > 0)
{
    If(frame_size > max_frame_size – (AV_FIXED_MEDIA_PACKET_HDR_SIZE + RTP_HEADER))
    {
        copy max_frame_size – (AV_FIXED_MEDIA_PACKET_HDR_SIZE + RTP_HEADER) to buffer;
        frame_size -= max_frame_size – (AV_FIXED_MEDIA_PACKET_HDR_SIZE + RTP_HEADER);
    }
```

```
        else
        {
            copy entire mp3_framse_size to buffer;
            frame_size -= mp3_frame_size;
        }
    }
```

RTP_HEADER is the 4 bytes header specified in RFC2250. max_frame_size is the maximum L2CAP frame size that can be used. This is signalled to the application in the AV_CONNECT_CFM message.

Please note that any royalty issue regarding MP3 is not within the scope of CSR Synergy Bluetooth or the CSR Synergy Bluetooth license.

For information about the MP3 frame format please refer to the following links.

http://www.mp3-tech.org/programmer/frame_header.html

http://blog.var.cc/blog/archive/2005/12/18/mp3-frame-length.html

http://www.oreilly.com/catalog/mp3/chapter/ch02.html

4 UART

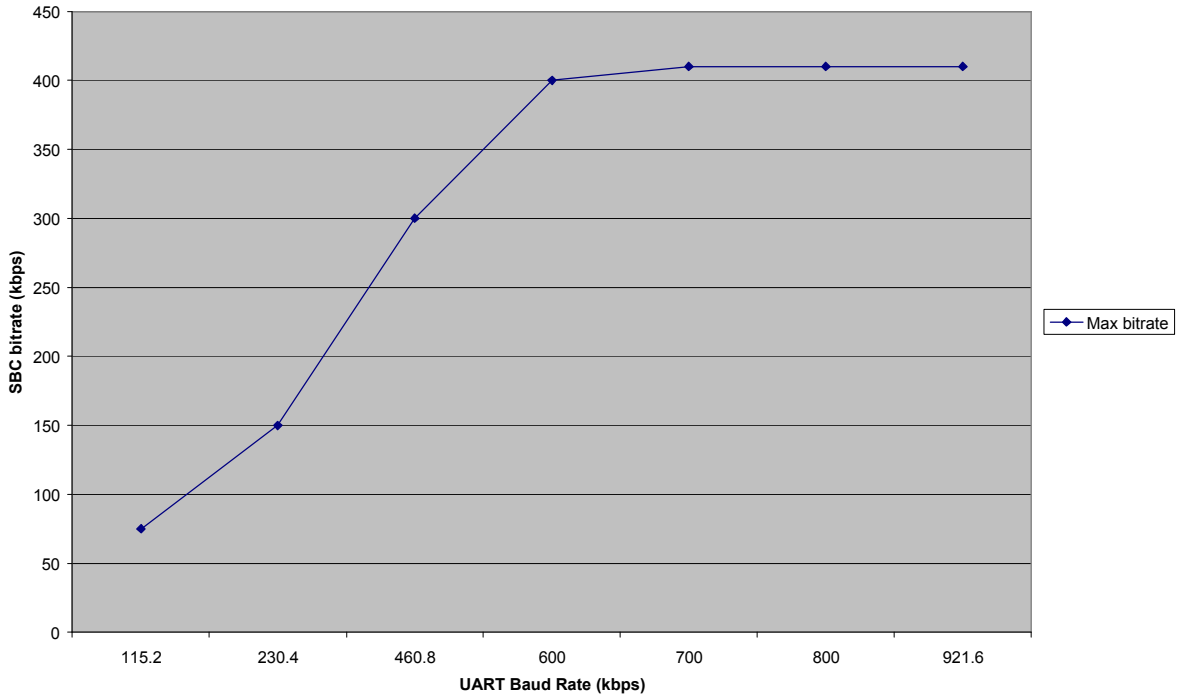
The UART connection must be reliable and have sufficiently high baud rate. The minimum baud rate is 460.8 kbps baud - a baud rate of 921.6 kbps baud is recommended. The higher baud rate is needed in order to allow for start/stop bits, Bluetooth protocol packet headers, byte stuffing in the SLIP protocol and the BCSP protocol itself (protocol headers and retransmission).

Below is a table showing the relationship between achievable SBC bitrate at different UART baud rates. The measurements were done using the A2DP demo application and a Samsung SBH-100 stereo headset using the following stream parameters:

Channel mode: SBC_JOINT_STEREO
Allocation method: SBC_METHOD_LOUDNESS
Sample frequency: 48 kHz
Subbands: 8
Blocks: 16

A USB to UART converter was used for achieving the high baudrates. The USB to UART converter was connected to a Casira with a BC2 module.

Baud rate (kbps)	Max SBC bitrate (kbps)
115.2	75
230.4	150
460.8	300
600.0	400
700.0	410
800.0	410
921.6	410



Please note that these measurements were done on a PC and the results achievable on an embedded platform are likely to be different. These measurements should only be used as an indication of the relationship between the UART speed on the platform and the maximum SBC bitrate achievable.

The maximum achievable SBC bitrate is determined as the bitrate where a continuous music stream is possible, i.e. before the music will start to break up.

Most headsets will put a limitation on the maximum bitpool (linear dependent on the SBC bitrate) that can be used, and this explains the upper limit reached at around 410 kbps SBC bitrate.

5 Architecture

The Bluetooth profile specification defines two separate profiles relevant for audio applications:

- The A2DP profile used for audio streaming
- AVRCP used for remote control; it is optional to support AVRCP

If AVRCP is supported, it is up to the AVRCP target (typically the audio source) to interface and map the control signals to a media player. The signals for handling for instance pause and play are received from the AVRCP profile layer but may need to be routed to the media player in the application layer to actually perform the wanted action.

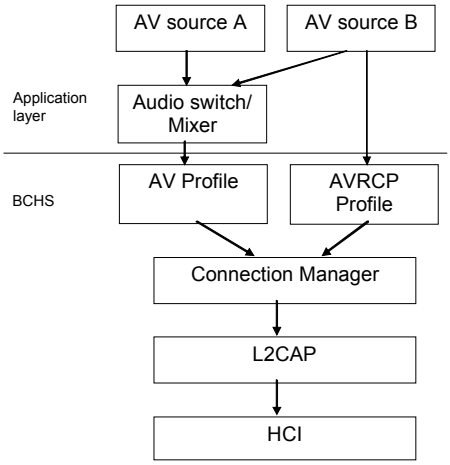


Figure 4: Reference application architecture

In the example shown in Figure 4, Audio source A only supports audio streaming, whereas Audio source B supports both audio streaming and remote control functions. This could for instance be in a system where B is a media player and A is responsible for internal sound generation, e.g. system sounds.

The CSR Synergy Bluetooth AV implementation support transfer of multiple simultaneous streams. However, most headsets on the market do not support this feature. Therefore, it is recommended to handle multiple streams on the source side, which can be handled in two different ways:

- Merge the audio streams into one stream before transmitting to the headset
- Time multiplexing the audio streams into one stream. In practise this can be accomplished by for instance by discarding or postponing the media player stream while playing short system sounds

The time multiplexing approach is only feasible if the additional audio stream is short audio notification, e.g. an alarm or similar. If it is required that it must be possible to stream more than one stream simultaneously, e.g. streaming music at the same time as hearing for instance gaming audio, the merging approach must be considered. If more streams should be routed to the AV headset simultaneously, a mixer must be introduced and it must be considered how to convert the audio streams to the same format for all streams, e.g. 44.1 kHz sample frequency.

6 Document References

Document	Reference
Bluetooth® Core Specification, Version no. 2.0, 2.1 and 3.0	[BT]

Terms and Definitions

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
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.