ZUMI

Bluetooth ANC Systems Interfacing

Introduction

The implementation of ANC in Bluetooth headsets is of particular interest because many Bluetooth chips provide sufficient processing capability to implement the required ANC filters. The ANC functionality can potentially be added to a product with no additional BOM cost.

ANC implementation requires a signal filter to transform the signal from a microphone to then be output from the speaker.

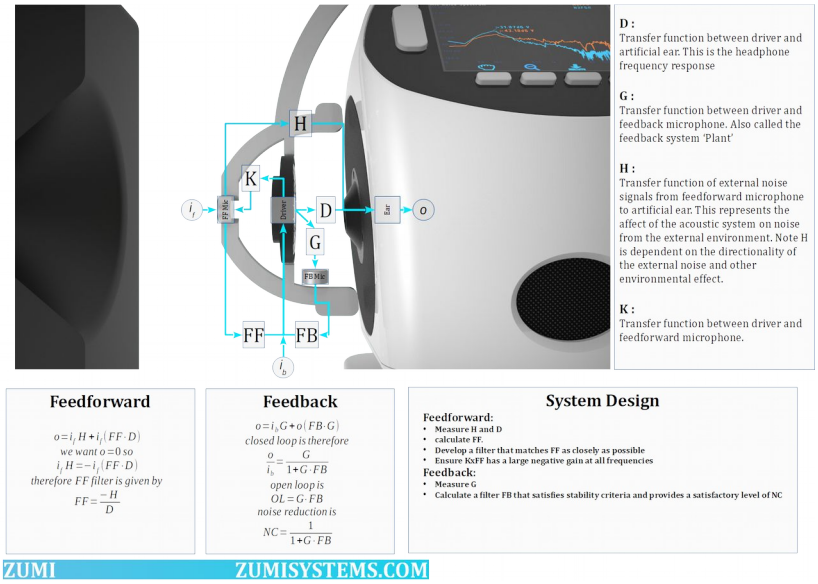
In the Bluetooth systems discussed here filtering is primarily done by digital means. The most common implementation of the digital filter is a cascade of bi-quad filters.

The filter is therefore defined entirely by a set a filter coefficients.

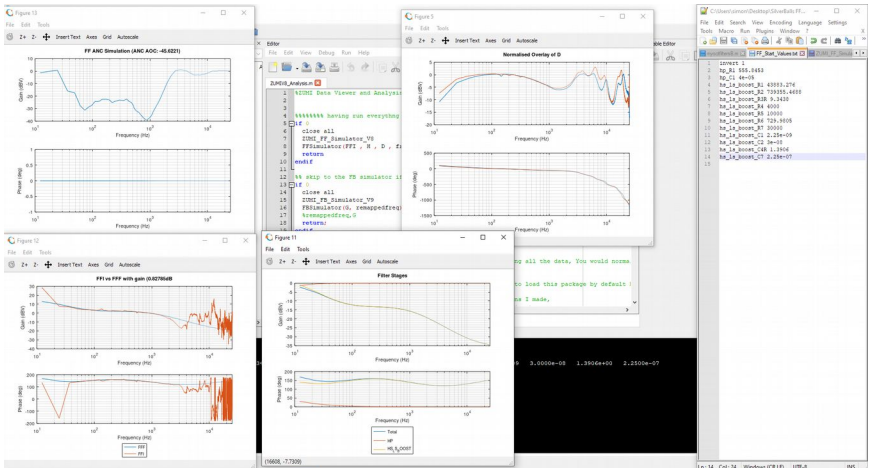
Our objective is to define methods for the ZUMI measurement instruments to GET and SET these filter coefficient sets via a Bluetooth SPP interface.

Background

The first stage of implementing ANC in a headphone is to obtain measurements of the physical acoustic system. These measurements ultimately result in a set of characteristic transfer functions for the headphone.



The measured data is then used in modelling and simulation software to compute the optimal filters



The computed filter values are then uploaded to the headphone via SPP and the gain trimming process is executed.

A gain trimming process is generally required for 100% of production units in order to compensate for component tolerances. The trimming process uses 2 algorithms, one for feedforward and one for feedback ANC.

Communications

For communications via serial interfaces (UART, SPI, SPP) ZUMI uses a simple framed command format by default.

uint8 StartFlag (0x2) // one byte

uint8 ReservedByte // always 0x0

uint8 CmdCode_upperbyte //

uint8 CmdCode_lowerbyte // two bytes for command code

uint8 Length_upperbyte

uint8 Length_lowerbyte // two bytes for payload length

uint8 Payload[Length] //

uint8 EndFlag (0x3) // one byte

Receiver would have to parse for the frame with the only check for a valid frame being that the value at (&StartFlag + Length) == EndFlag

The interpretation of the payload bytes is dependent on the command code. The payload may be any combination of integer, float or char.

ZUMI has a maximum buffer size of 65536 bytes. This means the complete message including framing bytes should not exceed 65536 bytes, hence the Payload length maximum is 65529 bytes.

A timeout counter of 2 seconds is started when a StartFlag is received. The timeout triggers if no bytes are received are within 2 seconds. Receiving any byte restarts the timeout counter.

Gain Trimming

Gain trimming is used for development and production of ANC headphones and earphones to set the optimal filter gain.

ANC systems can be feedforward and/or feedback topology.

The process of gain trimming will involve an iterative process of setting a desired gain level,

measuring the resulting ANC performance, computing an adjustment and setting a new gain level.

When the optimal (or acceptable) gain levels have been reached ZUMI will command the device to store the gain settings.

Gain Table

A gain table is required. The gain table is an array of gain values available.

The gain values should be dB gain.

The gain values are not required to be absolute voltage gain numbers, however the relative step size must be correct.

ZUMI computes a required gain adjustment as part of the the trimming process. The table is used to look-up the nearest gain level the device supports.

The gain table may be hard coded for vendor specific devices, or retrieved by a serial command.

Get Gain Levels

The device shall respond to requests for current gain level, returning

FF_L , FF_R, FB_L, FB_R

e.g.

<ZUMI>

0x2, // StartFlag

0x0,

GET_GAIN_REQUEST, // 2 bytes

0x0,

0x0, // zero length payload, its a command only

0x3, // EndFlag

<Device>

0x2, // StartFlag

0x0,

GET_GAIN_RESPONSE, // 2 bytes

0x0,

0x16, // 16 bytes payload, 4 vales of float32

{

4 x float32's

}

0x3, // EndFlag

Set Gain Levels

The device shall accept a command for setting the gain levels.

<ZUMI>

0x2, // StartFlag

0x0,

SET_GAIN_REQUEST, // 2 bytes

0x0,

0x16, // 16 bytes payload, 4 vales of float32

{

4 x float32's

}

0x3, // EndFlag

<Device>

0x2, // StartFlag

0x0,

SET_GAIN_RESPONSE, // 2 bytes

0x0,

0x1 // 1 byte payload,

0x1, // setting success or 0x0, // setting failed

0x3, // EndFlag

Get Device ID

The device shall provide a device ID code

<ZUMI>
0x2, // StartFlag
0x0,
GET_VENDORID_REQUEST, // 2 bytes
0x0,
0x0, // zero length payload, command only
0x3, // EndFlag
<Device>
0x2, // StartFlag
0x0,
GET_VENDORID_RESPONSE, // 2 bytes
0x0,
0x16, // 16 bytes payload, typically 16 char string for vendorID
{
char vendorid[16]
}
0x3, // EndFlag

Store Gain Levels

This command will instruct the device to 'make permanent' the current gain levels. Such
<ZUMI>
0x2, // StartFlag
0x0,
STORE_GAIN_REQUEST, // 2 bytes
0x0,
0x0, // zero length payload, command only
0x3, // EndFlag
<Device>
0x2, // StartFlag
0x0,

STORE_GAIN_RESPONSE, // 2 bytes

0x0,

0x1 // 1 byte payload,

0x1, // setting success or 0x0, // setting failed

0x3, // EndFlag

Mute Microphones

The device should be able to set the microphone signals to a mute state. The function may be redundant if the gain setting includes a negative 'huge number' gain.

<ZUMI>

0x2, // StartFlag

0x0,

MUTE_REQUEST, // 2 bytes

0x0,

0x1, // 1 byte payload, its a mute mask

0x3, // mute mask 0000 0011

0x3, // EndFlag

<Device>

0x2, // StartFlag

0x0,

MUTE_RESPONSE, // 2 bytes

0x0,

0x1 // 1 byte payload,

0x1, // setting success or 0x0, // setting failed

0x3, // EndFlag

Filter Coefficients

Most often digital filters used for ANC are an implementation of a series of cascaded digital Biquadratic filters.

In the general form each filter stage is defined by sets of coefficients (normally denoted as A0,A1,A2,B0,B1,B2) NOTE: Some implementations may normalise for A0 and B0.

ZUMI characterisation methods result in a target filter function in the form of an array of complex numbers. This result is essentially a discrete point transfer function H(s).

Computing coefficients directly from the transfer is not possible as there is no unique solution. However a curve fitting iterative algorithm could be used to obtain satisfactory results.

In the general case each of the ANC filters will have an independent set of the cascaded biquad filters, giving a table of filter coefficients i.e.

		FF_L	FF_R	FB_L	FB_R
Biquad Stage	Coefficeints				
A	A0	FF_L_A_A0	FF_R_A_A0	FB_L_A_A0	FB_R_A_A0
A	A1	FF_L_A_A1	FF_R_A_A1	FB_L_A_A1	FB_R_A_A1
A	A2	FF_L_A_A2	FF_R_A_A2	FB_L_A_A2	FB_R_A_A2
A	B0	FF_L_A_B0	FF_R_A_B0	FB_L_A_B0	FB_R_A_B0
A	B1	FF_L_A_B1	FF_R_A_B1	FB_L_A_B1	FB_R_A_B1
A	B2	FF_L_A_B2	FF_R_A_B2	FB_L_A_B2	FB_R_A_B2
B	A0	FF_L_B_A0	FF_R_B_A0	FB_L_B_A0	FB_R_B_A0
B	A1	FF_L_B_A1	FF_R_B_A1	FB_L_B_A1	FB_R_B_A1
B	A2	FF_L_B_A2	FF_R_B_A2	FB_L_B_A2	FB_R_B_A2
B	B0	FF_L_B_B0	FF_R_B_B0	FB_L_B_B0	FB_R_B_B0
B	B1	FF_L_B_B1	FF_R_B_B1	FB_L_B_B1	FB_R_B_B1
B	B2	FF_L_B_B2	FF_R_B_B2	FB_L_B_B2	FB_R_B_B2
C	A0	FF_L_C_A0	FF_R_C_A0	FB_L_C_A0	FB_R_C_A0
C	A1	FF_L_C_A1	FF_R_C_A1	FB_L_C_A1	FB_R_C_A1
C	A2	FF_L_C_A2	FF_R_C_A2	FB_L_C_A2	FB_R_C_A2
C	B0	FF_L_C_B0	FF_R_C_B0	FB_L_C_B0	FB_R_C_B0
C	B1	FF_L_C_B1	FF_R_C_B1	FB_L_C_B1	FB_R_C_B1
C	B2	FF_L_C_B2	FF_R_C_B2	FB_L_C_B2	FB_R_C_B2
...	...	...	...	...	...

Get Coefficients

The device should be able to output a list of filter coefficients for each filter.

<ZUMI>

0x2, // StartFlag

0x0,

GET_COEFFICIENT_REQUEST, // 2 bytes

0x0,

0x1, // 1 byte payload, its the filter ID number

FF_L_COEFFICIENTS_ID, // e.g. FF_L = 0x1, FF_R = 0x2, FB_L = 0x3, FB_R = 0x4

0x3, // EndFlag

<Device>

0x2, // StartFlag

0x0,

GET_COEFFICIENT_RESPONSE, // 2 bytes

0x0

0x65 // 101 byte payload,

{

0x1 , // return the filter ID

0x4, // e.g. the number of filter stages

float32 coeffs[4][6], // 4 stages, each with 6 coefficients as 32 bit floats

}

0x3, // EndFlag

Set Coefficients

The device should set the filter coefficients for each filter upon request

<ZUMI>

0x2, // StartFlag

0x0,

SET_COEFFICIENT_REQUEST, // 2 bytes

0x0,

0x65 // 101 byte payload,
{
0x1 , // return the filter ID
0x4, // e.g. the number of filter stages
float32 coeffs[4][6], // 4 stages, each with 6 coefficients as 32 bit floats
}
0x3, // EndFlag
<Device>
0x2, // StartFlag
0x0,
SET_COEFFICIENT_RESPONSE, // 2 bytes
0x0,
0x1 // 1 byte payload,
0x1, // setting success or 0x0, // setting failed
0x3, // EndFlag

Store Coefficients
The device should be able to permanently store the 'current' coefficients.
<ZUMI>
0x2, // StartFlag
0x0,
STORE_COEFFICIENT_REQUEST, // 2 bytes
0x0,
0x0 // zero byte payload,
0x3, // EndFlag
<Device>
0x2, // StartFlag
0x0,
STORE_COEFFICIENT_RESPONSE, // 2 bytes
0x0,
0x1 // 1 byte payload,

0x1, // setting success or 0x0, // setting failed

0x3, // EndFlag