Bowers & Wilkins

Dear _____,

Thank you for accepting the Bowers & Wilkins DSP Coding Challenge.

This exercise is designed to test some of the day-to-day technical skills required by an Audio DSP engineer, as well as giving an opportunity to demonstrate your creative thinking and communication abilities.

The DSP Coding Challenge 2025:

- **Part 1: Coding Exercise**
  Audio algorithm design & implementation.

- **Part 2: Questions**
  Demonstration of subject knowledge, creative engineering, and communication skills.

The exercise is designed to take a few hours to complete - We thank you for your time and effort. But ask that you do NOT spend a lifetime on the project.

While a technical test for interview candidates, we hope the task is also interesting to complete. Please remember that the purpose is to demonstrate your current coding-style / ability level – some gaps in knowledge are acceptable – if you encounter problems please move on as there are plenty of other elements where you can impress us.

If you can't get part of the project working, just comment out the offending part (don't delete it) and write a brief description of what's wrong / what should be happening.

If you get stuck or have questions, please email the address below and we shall try to get you moving again asap.

Good luck. We look forward to seeing what you come up with.

Keith Ovenden
Lead DSP Integration Engineer
keith.ovenden@soundunited.com

Part 1: Coding Exercise

Please create a C/C++ program (header and source files) to apply a FILTER and VOLUME CONTROL to a STEREO audio stream.

Please comment and annotate your code as you see fit.

Important Information:

Audio Samplerate:	48kHz	
Audio 'Interrupt' Packet Size:	64 Samples	
Filter Type:	YOUR CHOICE	
Filter Order:	2nd-Order	
Freq:	YOUR CHOICE	
Q:	YOUR CHOICE	
Gain:	YOUR CHOICE	(If relevant to filter type)
Volume Fader Level:	YOUR CHOICE	
Output Buffer Audio Format:	2 channels interleaved	

Your program should contain:

i. **Main Function.**
- Call functions to set up the system
- Call audio function to process ALL the generated data.

ii. **Initialization Function.**
- Initialise your filter.
- Set up your Volume Fader.
- Synthesize TWO channels of audio data into memory.
  - **1024** samples for each source.
  - CH1 = 100Hz Sinewave - Volume = -6dBFS
  - CH2 = 1kHz Sinewave - Volume = -10dBFS

iii. **Control Functions.**
- Set Filter Function - Pass in 'raw' parameters & generate coefficients.
- Set Volume Function - Pass in a level in dB, set correct gain.

iv. **Audio-Processing Function**
- Called from main() to simulate an audio interrupt.
- Passed pointer(s) to the correct 'source' audio data.
- Passed a pointer to the correct audio data 'destination.'
- Process **ONE** full packet of samples with each call.

Part 2: Questions

Note: No need to code, just briefly explain in words.

Please write answers in a suitable file format and send along with your source files from Part-1 in a zip file to the email address provided on Page 1.

1) Describe your filter design.
What type of implementation have you used?
Where have the coefficients come from?

I implemented a second-order low-pass filter (biquad) with a cutoff frequency of 1135 Hz and a quality factor of 1.493.

I applied the bilinear transformation to the general analog filter prototype to derive the coefficients for my digital filter. I selected fc and Q to achieve a resonance peak of approximately +4dB around 1kHz.

My original goal was to boost the 1000Hz sine component to match the level of the 100Hz component.

2) If the user moves the Volume Fader from 0dB to -10dB, what should be done to change volume without audible clicks?

To avoid audible clicks, we can gradually reduce the volume so that the signal transitions are smooth and without abrupt changes.

3) If the user moves the Volume Fader to +10dB, what would happen to the test signal at the output?

If the signal amplitude exceeds a certain threshold (such as a voltage limit), it may experience clipping.

The signal is therefore truncated, resulting in distortion.

4) For High-Pass/Low-Pass filters, what happens to the frequency response when the Q value rises above 0.707?

The frequency response will show a reasonance peak for Q > 0.707

5) How could you achieve a steeper filter roll-off?

The most effective method is to increase the filter's order. We can, for example, cascade several biquad filters, which is better than a single filter of higher order.

6) What would you need to do to the algorithm to (dynamically) change the Samplerate to 96kHz?

In my implementation, I can adjust the normalized angular frequency w0 according to the new sampling rate and recompute my filter coefficients.

I would also upsample my input signal by a factor of 2 by inserting zeros

7) Describe THREE ways in which you could extend your program to add some more functionality/flexibility.

ONE:

Display both the input and output signals in a graphical format, allowing users to observe how the filter affects the waveform. Optionally also integrate real-time frequency response visualization.
This could be done by extracting the signals and different information from this program.

TWO:

Add a wider range of filter types (high/band-pass, notch,shelf...) and allow users to combine multiple filters for more modulation and fune-tuning.

THREE:

Extend the program to support various signal types other than sine waves, such as white noise, square waves, and sawtooth waves. Also, incorporate different type of audio synthesis for even more flexibility.