DSP implementation of surround speaker virtualizer for sound bar systems

Nives Kaprocki, Jelena Kovačević, *Member, IEEE*, Robert Pečkai Kovač, and Miroslav Malko

***Abstract*—This paper describes one solution for an implementation of the surround speaker virtualization module for a digital signal processor in a sound bar system. The main problem in this system, which consists of a decoder, a renderer and a post-processing module, is complex processing of sound in real-time. In order to achieve real-time execution of these operations and offer customers an affordable product, sound bar system is implemented on an embedded platform. A detailed overview of the system architecture and available resources is given, as the development of modules in an embedded system requires optimal resource usage. Furthermore, numerous challenges and constraints of real-time system programming are presented through the implementation of surround virtualization module, an integral part of the post-processing block. The merits of the proposed architecture are fortified with the measured usage of limited processor resources, and evaluation results confirm that real-time execution of the proposed solution is achieved.**

***Key words* — audio processing, DSP, real-time, sound bar, surround sound, virtualization.**

I. Introduction

When high-definition television (HDTV) was introduced, displays became thinner and built-in speakers became smaller and weaker. Manufacturers promoted sound bar products as a solution to improve the audio so that it matched the quality of the video. Although in the beginning sound bars had only replaced the sound missing from the on-board speakers, its role soon became delivery of high quality audio experience [1].

In order to provide a deeper sense of immersion, many improvements in the sound bar design and performance have been made. First, speakers which direct sound upward in order to recreate overhead sounds have been added. Separate speakers above the listener needed to be used in the past to achieve the same effect. Furthermore, advanced audio decoders which are capable of processing content encoded for different formats have been employed. Lastly, advanced audio processing is used to correct audio challenges, enhance the channel based content and provide surround sound without the surround multiple-speaker configuration.

Various kinds of speaker set-ups are classified with two or three digits separated by a decimal point (2.1, 5.1, 7.1, 5.1.2, 7.1.2 etc.) [2]. The first digit shows the number of primary channels, each of which is reproduced on a single speaker [3] (these speakers are capable of handling the frequency range from 100Hz to 22kHz), the second (decimal digit) refers to the presence of LFE (Low Frequency Effect), which is reproduced on a subwoofer, while the third refers to the presence of the top channels, reproduced on speakers above the listener. When arranged around the room, these speakers produce a surround effect (Fig. 1). The sound bar, equipped with a digital signal processor that can perform surround speaker virtualization [5], offers consumers a good alternative for the expensive and cumbersome multiple speaker configuration, while not compromising the quality of the surround effect.

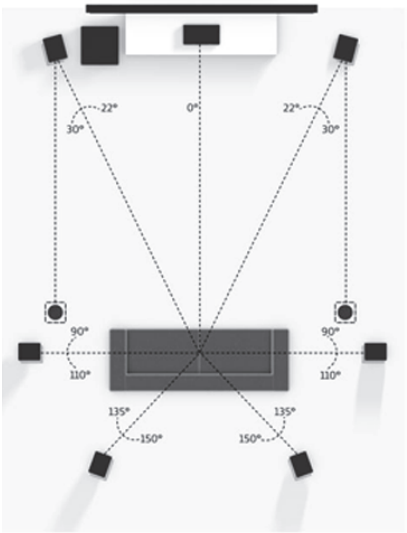

Fig. 1. 7.1.2 speaker configuration [4].

This paper describes the architecture of the sound bar system with a Cirrus Logic digital signal processor, with emphasis on the implementation of the surround speaker virtualization module. The implementation uses a combination of advanced head-related transfer functions

Paper received May 5, 2017; revised August 19, 2017; accepted September 25, 2017. Date of publication December 25, 2017. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Branimir Reljin.

This paper is a revised and expanded version of the paper presented at the 24th Telecommunications Forum TELFOR 2016 [13].

This work was partially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia under Grant TR-32029.

Nives Kaprocki, RT-RK Institute for Computer Based Systems LLC Novi Sad, Serbia (e-mail: nives.kaprocki@rt-rk.com).

Jelena Kovačević, Faculty of Technical Sciences Novi Sad, Serbia (e-mail: jelena.kovacevic@rt-rk.com).

Robert Pečkai Kovač, RT-RK Institute for Computer Based Systems LLC Novi Sad, Serbia (e-mail: robert.peckai-kovac@rt-rk.com).

Miroslav Malko, RT-RK Institute for Computer Based Systems LLC Novi Sad, Serbia (e-mail: miroslav.malko@rt-rk.com).

(HRTFs) [6], [7] and crosstalk cancellation [8], methods which in basic form are used for the simulation of surround sound in classical audio devices, audio/video receivers and headphones. The main issue is enabling real-time sound processing of the surround speaker virtualizer, integrated with other audio post-processing modules (dialogue enhancer, downmixer, intelligent equalizer, etc.). For this array of complex audio processing to work continuously and consistently, the implementation has to enable real-time computing by efficient handling of input data [9]. When the processing is successfully done within the real-time constraints, the virtualized sound reproduced on the sound bar placed in front of the listener reaches the listener from all directions thus providing surround effect.

II. System Architecture

Hardware architecture of the sound bar system (Fig. 2) consists of these main components:

- Quad-Core 32 bit Digital signal processor (DSP)
- Microcontroller unit (MCU)
- Audio digital-to-analog converters (DAC) and analog-to-digital converters (ADC)
- Flash memory
- SDRAM memory

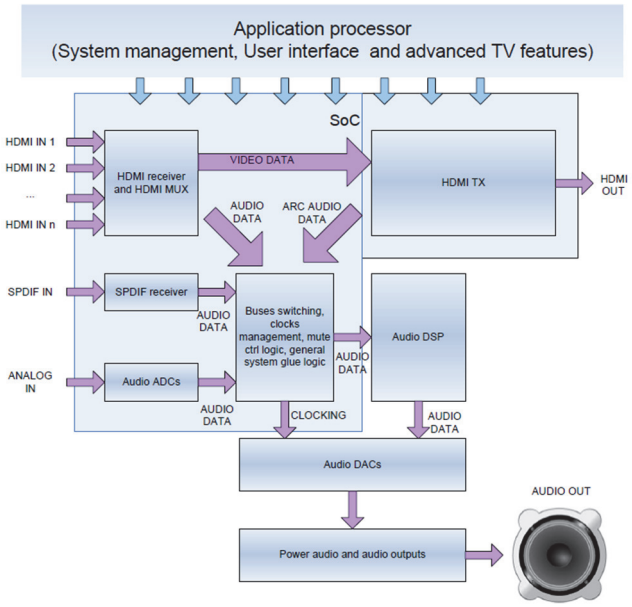

Fig. 2. Block diagram of the sound bar system.

Sound bar is connected with external devices through HDMI (max. 24 Mbps) or SPDIF (max. 3 Mbps) interfaces, while the received audio content is transferred to the DSP using I2S (max. 24 Mbps) interface. DSP and Flash Memory are interconnected using SPI (6 Mbps), which loads firmware modules and saves system configuration parameters. Finally, I2C (100 Mbps) interface is used for system configuration and control.

The central component in the sound bar system is a quad-core digital signal processor, which is controlled by a general purpose processor (MCU). MCU is also responsible for system management and user interface (UI) events. MCU handles digital input data through an HDMI or SPDIF receiver, while an analog input is converted into a digital input with Audio ADCs. When the DSP receives an input stream or a change of input stream happens, it sends messages to the MCU.

Fig. 3 shows communication between DSP and the MCU. Depending on the input stream, the DSP switches between a decode state and a waiting state. Autodetect and Audio Configuration Change Notification (ACCN) messages are reported in the decode state, while a "Silence" message is reported when the DSP returns back to a stream wait state. While the first autodetect message contains only information about the input stream type, ACCN message contains stream specific details. These details include information such as the number of channels, output channel mask and stream's sample rate. When this message is parsed, the MCU loads a suitable configuration file from the Flash Memory into the DSP. The DSP acknowledges this by re-sending the ACCN message. The re-sent ACCN message is parsed and individual run-time configurable commands are sent. Acquired audio content is then processed by the DSP and played on the speakers.

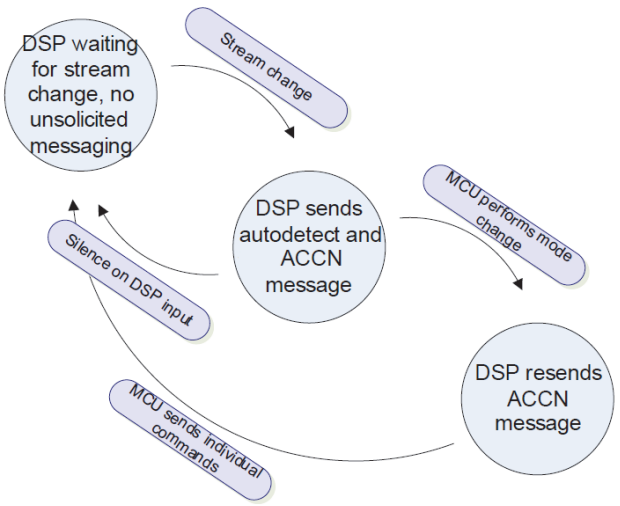

Fig. 3. DSP to Microcontroller Messaging Diagram.

Cirrus Logic CS49844 DSP processor consists of four 32 bit DSP cores, which are based on an advanced Harvard architecture [10], and have separate X and Y data and P code memory spaces. Each DSP core can spend a maximum of 300 MIPS for their processing and use 60k of X and Y data memories and 60k of P code memory (Table 1). The data between the cores is sent through a SDRAM memory with a size of 64k words. The SDRAM runs at 133MHz clock frequency and has a 64Mb data flow in one transfer.

Table 1: Available processor memory.

Memory type	Core A	Core B	Core C	Core D
X,Y,P	60kWord SRAM	60kWord SRAM	60kWord SRAM	60kWord SRAM

Each core is a high-performance, 32-bit, user-programmable, fixed-point DSP [11] that is capable of

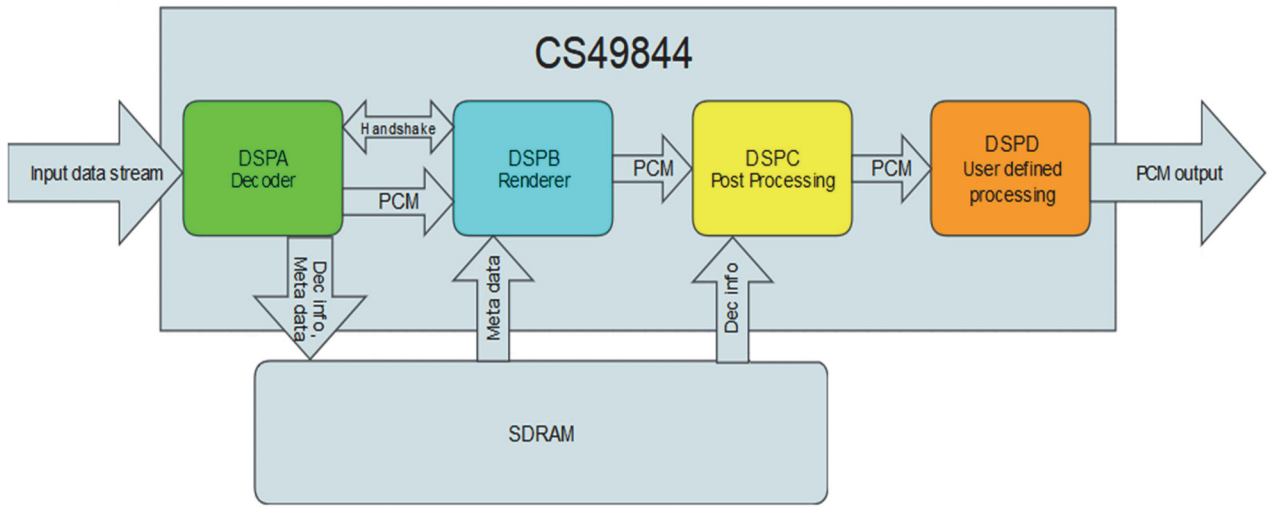

Fig. 4. Allocation of modules in the sound bar system.

performing two memory access control operations per clock cycle. One core has eight 72-bit accumulators, four X and four Y data registers, and 12 index registers.

The whole sound bar system consists of three main modules: decoder, renderer and post-processing module. Fig. 4 shows which module is allocated to which DSP core. Decoder is placed on the first DSPA core. It receives a compressed data stream from HDMI or SPDIF serial ports and generates PCM samples. Various decoders can be used in these systems, such as Dolby Digital and Dolby Digital Plus. Autodetection and MCU control which decoder will be loaded into the first core. Decoded data is stored in the internal PCM array, from which it is copied to the input-output (IO) buffer. Samples cannot be stored directly in the IO buffer, as the amount of decoded data is larger than the buffer's size. The size of the internal PCM double buffer depends on the decoder and the sample rate, and varies between 80 and 3840 Words per channel, while the size of the IO circular buffer is 128 Words per channel.

Output PCM samples from the decoder are input samples for the renderer. Renderer arranges objects, virtual sources used for spatial sound reproduction, depending on the received metadata and the output speaker configuration. Initialization of this module must be performed before rendering of the input data. The second core notifies the decoder when the initialization is done, and the decoder starts to send metadata into SDRAM memory. Metadata includes spatial coordinates, loudness and other indicators that enable adaptation of sound objects to the resulting speaker configuration [12]. As soon as the metadata is received, the processing begins – removal of the objects from the input channel content and their addition to the output speaker content. Finally, the third DSPC core is used for audio post-processing, which includes surround speaker virtualization. This core also requires certain data from the decoder, such as the sample rate and output speaker configuration, whose transfer is handled through a SDRAM.

III. Algorithm

The algorithm for virtualization of surround speakers generates a Hybrid complex quadrature mirror filter bank (HCQMF) structure with a buffer holding audio channels of a stream in 5.1 or 5.1.2 format from an input HCQMF structure with channels in arbitrary formats. Additionally, the algorithm amplifies the surround channels, in order to create a more realistic surround effect. Fig. 5 shows a block diagram of the algorithm.

A. Adjustment of channels' format to 5.1 or 5.1.2

The input parameter of this processing block is a HCQMF structure, which contains configuration of the buffer with audio channels (e.g. number of channels, presence of height channels), as well as the buffer itself.

When the input buffer does not contain top surround channels and the output format is 5.1.2, these two channels are added and filled with zeroes. If the number of primary channels in input buffer is higher than it is required, downmixing is performed. Lastly, in case of an input format that has two additional top surround channels which the 5.1 output format does not need, these channels are omitted.

The result of the processing is a modified HCQMF structure with either a 5.1.2 or a 5.1 buffer, which the virtualization processing block expects as an input parameter.

B. Generation of coefficients

Three input parameters are required for this processing, which is based on the HRTF, as this transfer function depends on frequency and two spatial variables - $H(f, \theta, \varphi)$. The first, center frequency of each virtualization band normalized to the sample rate. The second, the angle from the listener (relative to the forward facing axis) to where the speaker should sound like it is coming from. The third, the angle from the listener (relative to the forward facing axis) to where the physical speaker is actually located. When distance from the head to the source is greater than 1m, the

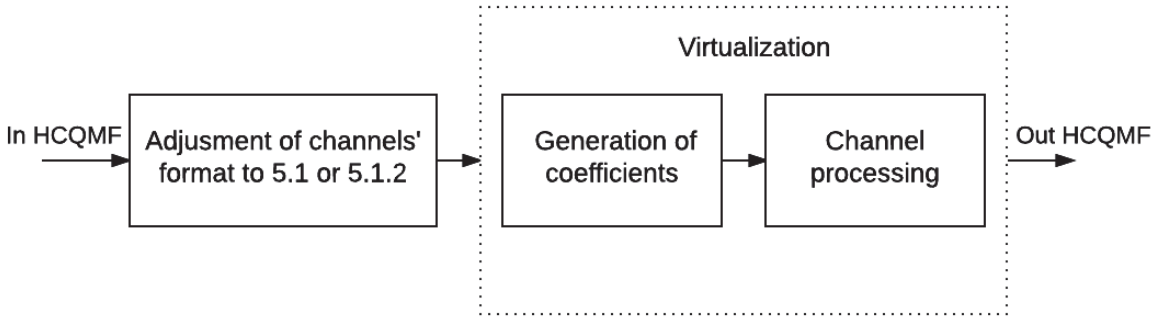

Fig. 5. Block diagram of the algorithm.

HRTF attenuates inversely with range. As this is the case in sound bars, distance is not provided as one of the input parameters.

Processing which also takes into account the time difference of the arrival of sound waves at left and right ear canals and cancels the sound that arrives from one ear to another, generates two sets of 77 coefficients, both stored as interleaved real and imaginary pairs.

C. Channel processing

Finally, the channels are processed and virtualized using two sets of complex coefficients – direct and cross-mixed. The first set of complex multipliers is applied directly to the left and right channels when they are mixed into the left and right output channels, whereas the second set is applied to the left and right channels when they are mixed into the right and left output channels. Virtualization is performed on forwarded left and right surround channels and, in case of 5.1.2 output configuration, on left and right top surround channels.

Providing that the supplied coefficients are normalized, surround speaker virtualization can amplify the sound for the maximal value of 5.7 dB. Result of this processing is a pair(s) of channels that when played on the sound bar system in front of the listener, produces sound that simulates its theatrical location.

IV. Implementation

Surround virtualization algorithm is implemented using the Cirrus Logic framework. Cirrus Logic framework is a system programming support of the processor, which simplifies the development of the application. It introduces the idea and methodology of object oriented programming into the assembly code [13]. The core of the framework is a simple real-time operating system (OS), which calls module routines in a predefined order.

When the IO buffer receives 16 new PCM samples for every channel (maximal number of input channels is 10), the Block routine is called and these samples are stored in a buffer. As the flow of new samples is continuous and processing is done in real-time, a double (Ping-pong) buffer must be used. The handling and storing of the samples is presented in Fig. 6. One half of the buffer, labeled as the Foreground buffer in the figure, holds processed samples, which are exchanged with new samples from the IO buffer in the Block routine. While the Foreground buffer is being filled with new samples, the other half, labeled as the Background buffer, is being processed in the Background routine. When the processing is finished and 256 new samples are loaded for all the existing channels, the half of the Ping-pong buffer with new samples becomes the Background buffer, whereas the half with processed samples becomes the Foreground buffer.

The aforementioned double buffer is not the buffer that the surround module expects as its input parameter. Using hybrid synthesis and complex quadrature mirror filter, the buffer with 256 samples for every channel is transformed into a new buffer with four blocks of 77 complex samples for every channel. After the processing is done, the inverse transformation is performed, and the processed samples are arranged in the double buffer, ready to be forwarded to the IO buffer.

In order for the module execution to be successful in real-time, the processing has to be done before the Foreground buffer is filled with new samples. As the surround virtualization is not the only processing being done inside the Background routine, this can be very challenging.

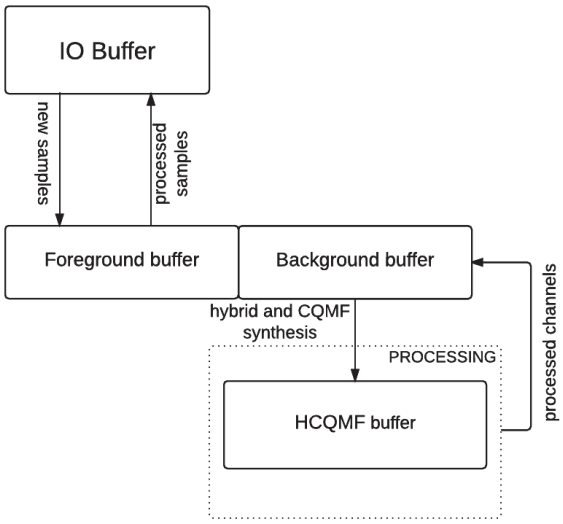

Fig. 6. Block diagram of the implementation.

For this reason, the system is implemented in assembly language, which provides full control of the processor's resources. Implementation starts with the analysis of the algorithm written in programming language C. After the main data structures and key functions are spotted, code

optimization and adjustment to the platform is done [14]. Optimization by reorganization of data refers to using registers and accumulators instead of variables for faster and more efficient access to temporary data. Also, large structures which describe the state of the application or its processing modules are created as global instances so that they could be accessed directly in functions.

One of the main hardware expansions in the digital signal processor is the Parallel Address Generation Unit. The unit consists of twelve 16-bit index registers and twelve 16-bit modulo-offset registers, which enable linear, reverse binary and modulo addressing modes. Modulo addressing is used to implement circular buffers and reverse binary addressing is used for Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT) algorithms.

As the algorithm includes lots of filters, most processing time is saved by reducing the number of instructions in programming loops. The architecture of the DSP enables a high level of instruction parallelism (multiplication, multiply and accumulate, bitwise operations and their combination), which is used as the main optimization technique in loops. Additionally, primitive operations can be performed to approximate divide, power, and square root functions, which normally would spend up to ten times more processing time. Implementation of all the aforementioned optimization techniques during development of the module on the target platform achieves successful real-time sound processing.

V. Results

The assessment of resource consumption of the audio post-processing is done on an one-core simulator. The evaluation of the surround virtualizer implementation is presented through its memory consumption and processor time usage, which, together with resource consumption of other post-processing modules, must not exceed the available amount of DSP core resources. Memory consumption is shown with the number of used words – 32 bit memory locations. Processor time usage, which is measured in a number of spent cycles by the development environment, is converted into million instructions per second with the following equation:

$$MIPS = \frac{cycle\_number * \frac{Fs}{BLOCK\_SIZE}}{1000000}. \quad (1)$$

Parameter Fs is the sampling frequency, number of samples obtained in one second, which equals 48kHz in the project. BLOCK_SIZE is the size of the processing block, which equals 256 samples.

Consumption of processor memory by the entire audio post-processing is shown in Table 2. The first two columns provide information on occupied data memory. The last column shows the amount of code memory used for audio post-processing. The amounts are within the limits of the available DSP core memories, given in Table 1.

Consumption of processor memory by the surround virtualization module is shown in Table 3. The first two columns provide information on data memory consumption. The last column shows the amount of code memory used in the surround virtualization module. It is easy to calculate that virtualization processing uses 17.9%, and 14.3% of the X and Y data memory respectively, and 3.6% of the overall code memory. Although all needed processes do not exceed the memory consumption to obstruct the real-time processing, this comparison easily shows that virtualization module does not add much to overall data usage, and thus it is also optimized.

Table 2: Used overall memory (Number of words).

X memory	Y memory	P memory
10298	11109	32992

Table 3: Used memory (Number of words).

X memory	Y memory	P memory
1841	1590	1208

Worst-case MIPS consumption of the entire audio post-processing is 156 MIPS, which was acquired by controlled execution of audio processing over different audio streams, with ten different input configurations per each stream. Since 300 MIPS can be spent on this processing, the conditions for real-time playback are satisfied.

As the integration of surround virtualization module into the audio enhancement system was successful, and the overall audio post-processing resource consumption is within the real-time constraints, further analysis of the resource consumption by the module can be performed.

Table 4 presents average MIPS consumptions, measured by controlled execution of the virtualization module for different input audio streams and output configurations.

Table 4: Consumption of processor time for different input audio streams.

Input stream format	Output configuration	MIPS
7.1.2	5.1.2	8.63
7.1	5.1.2	7.79
5.1.2	5.1.2	7.79
5.1	5.1.2	7.89
7.1.2	5.1	5.17
7.1	5.1	4.39
5.1.2	5.1	4.27
5.1	5.1	3.90

The consumption of processor time depends on the format of the input stream and the output configuration. The processing where the output configuration is 5.1.2 requires virtualization of both left and right surround channels, and left and right top surround channels. On the other hand, as the 5.1 configuration does not contain top surround channels, only left and right surround channels are virtualized.

A larger number of virtualized channels leads to a considerably more demanding processing, and higher consumption of processor time. The initial adjustment of the output configuration also affects the processor time consumption. For this reason the virtualization of the stream with 7.1.2 input format and 5.1.2 output configuration is most demanding. First, downmixing must be performed in

the adjustment phase, and then both the surround and top surround channels must be virtualized.

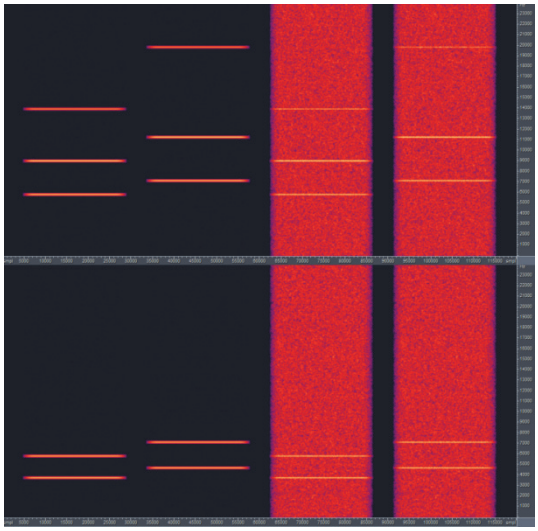

Fig. 7. Spectral view of the Left Surround and Left Back channel of the original stream.

Fig.7 shows a spectral view of the Left Surround and Left Back channels of a 7.1 input stream. As the output configuration is 5.1.2, seven primary channels are downmixed to five, which means that all primary channels are untouched except for the Left and Right Surround channels. These channels are calculated as the sum of Left Surround and Left Back channels of the original stream.

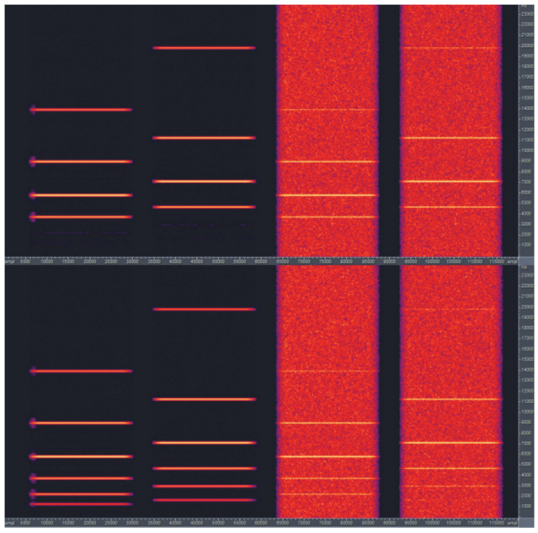

Fig. 8. Spectral view of the Left Surround channel before and after surround virtualization.

The top image in Fig. 8 shows a spectral view of the Left Surround channel after downmixing of the input stream. Downmixing is followed by surround virtualization of both Surround and Top Surround channels. The bottom image of Fig. 8 shows a spectral view of the virtualized Left Surround channel. Additional spectral components in the Left Surround channel appear as a result of the decorrelation process, when left and right channel are mixed.

VI. Conclusion

A surround speaker module for the Cirrus Logic digital signal processor in sound bar systems was presented in this paper. This module uses methods for simulation of surround sound based on the head related transfer function and interaural crosstalk cancellation, which re-create both surround and overhead sounds. Sound bars, equipped with additional speakers and described real-time sound processing, are able to provide consumers with a full, 360º surround sound.

The biggest challenge in this solution was implementation of the whole sound bar system on the embedded platform with a limited amount of memory and processing time. As the new generation of audio decoders requires more resources, post-processing algorithms, including virtualization of the multi-channel audio streams, have to be implemented using a very limited amount of resources. These limitations are overcome by realization of the algorithm in the low level programming language and optimization of the implemented code.

Future improvements in this module can be primarily done by a higher level of parallelism and more extensive use of the advanced DSP Harvard architecture, with a separate program memory and two separate data memories.

References

[1] R. Bleidt, A. Borsum, H. Fuchs, S. Merrill Weiss, "Object-Based Audio: Opportunities for Improved Listening Experience and Increased Listener Involvement," *SMPTE*, 2014.

[2] Multichannel sound technology in home and broadcasting applications, International Telecommunication Union Report, 2009.

[3] Guillaume Potard, "3D-audio object oriented coding," University of Wollongong, 2006.

[4] Demo World, http://www.demo-world.eu/infos/, 25.04.2017.

[5] B. C. O'Toole, M. Gorzel, I. J. Kelly, "Virtual 5.1 surround sound localization using head-tracking devices," *ISSC/CIICT*, 2014.

[6] P. Georgiou, C. Kyriakakis, "Modeling of head related transfer functions for immersive audio using a state-space approach," *Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers*, 1999.

[7] Jakes Bejoy "Virtual surround sound implementation using deccorrelation filters and HRTF," Center for computer research in music and acoustics, Stanford University, 2009.

[8] S. Cecchi, A. Primavera, M. Virgulti, "An efficient implementation of acoustic crosstalk cancellation for 3D audio rendering," *ChinaSIP*, 2014.

[9] I. Kalmykov, K. Katkov, N. Olegovich, A. Sarkisov, A. Makarova, "Parallel modular technologies in digital signal processing," *Life Science Journal*, 2014.

[10] V. Kovacevic, M. Popovic, M. Temerinac, N. Teslic, "Arhitektura i algoritmi digitalnih signal procesora 1," FTN, 2005, pp. 24.

[11] CS49844 Data Sheet, Cirrus Logic, Inc., May 2012.

[12] R. Oldfield, B. Shirley, J. Spille, "An object-based audio system for interactive broadcasting,2 *AES 137th Convention,* Los Angeles USA, 2014.

[13] N. Kaprocki, J. Kovačević, R. PečkaiKovač, M. Malko, "DSP implementation of the surround speaker virtualizer for sound bar systems," *24th Telecommunications Forum (TELFOR)*, 2016.

[14] M. Djukić, N. Četić, J. Kovačević, M. Popović, "A C compiler based methodology for implementing audio DSP applications on a class of Embedded Systems," *IEEE International Symposium on Consumer Electronics*, 2008.