A computationally efficient DAB bit-stream processor.

Renan Kazazoglu[1]
Suleyman S. Demirsoy[1]
Izzet Kale[1,2]
Richard C.S. Morling[1]

[1] School of Informatics, University of Westminster
[2] Applied DSP and VLSI Research Centre, Eastern Mediterranean University

A Computationally Efficient DAB Bit-Stream Processor

Renan Kazazoglu[*], Suleyman S. Demirsoy[*], Izzet Kale[*+] and Richard C.S. Morling[*]
[*]Applied DSP and VLSI Research Group
University of Westminster
London, UK
[+]Applied DSP and VLSI Research Centre
Eastern Mediterranean University
Gazimagosa, Mersin 10, Turkiye
RenanKazazoglu@hotmail.com, SDemirsoy@hotmail.com, kalei@wmin.ac.uk, morlingd@wmin.ac.uk

Abstract— **This paper describes an MPEG (Moving Pictures Expert Group) Audio Layer II – LFE (Lower Frequency Extension) bit-stream processor targeting DAB (Digital Audio Broadcasting) receivers that will handle the decoding of the frames in a computationally efficient manner to provide a synthesis sub-band filter with the reconstructed sub-band samples. Focus is given to the frequency sample reconstruction part, which handles the re-quantization and re-scaling of the samples once the necessary information is extracted from the frame. The comparison to a direct implementation of the frequency sample reconstruction block is carried out to prove increased computational efficiency.**

I. Introduction

Digital Audio Broadcasting (DAB) is a system designed by the European Telecommunication Standard (ETS) to transmit high quality digital audio and programme related data services using the terrestrial and satellite transmitters and cable networks in the Very High Frequency (VHF) and Ultra High Frequency (UHF) bands [1]. As raw digital audio has a high bandwidth, a method of data reduction and packing has to be applied to the sampled audio data before it can be transmitted over the medium. To this end, DAB uses the MPEG Audio Layer II – LFE coding algorithm [1], [2]. This coding scheme, depending on the results of the psycho-acoustic analysis performed on the data, allocates bits to the audio samples and packs these samples along with the ancillary data into frames to be transmitted over the medium.

The capability of providing CD quality audio and data transmission services is the main advantage of DAB over FM. However, the increased quality of audio and the added services result in relatively complex receiver architectures, often consuming more power. The aim of this paper is to describe a computationally efficient DAB bit-stream processor targeting mobile applications, as current commercially available DAB receivers are limited to standalone units or in-car kits.

A DAB decoder can be divided into three basic blocks, as it is seen in Fig. 1. The DAB bit-stream processor described in this paper deals with the bit-stream unpacking and frequency sample reconstruction blocks, with emphasis on increasing the computational efficiency of the frequency sample reconstruction block, leaving the synthesis sub-band filter block for future study. Section II of this paper will detail the architecture of the bit-stream processor and elaborate on the techniques utilized in the frequency sample reconstruction block in order to increase its computational efficiency. Further power reductions will be proposed in Section III by replacing the combinational multipliers with Reconfigurable Multiplier Blocks (ReMB) [3].

II. Architecture of the DAB Bit-Stream Processor

This section details the architecture of the designed DAB bit-stream processor. The first part will briefly discuss the structure and operation of some of the important sub-blocks of the bit-stream unpacking block in order to further clarify the improvements brought to the frequency sample reconstruction block. This will be followed by the details of the techniques utilized in the frequency sample reconstruction block to increase its computational efficiency. In order to render the overall design power efficient, the bit-stream processor was designed to work bit-serially, minimizing the dynamic power dissipation of the processor. The format of the DAB bit-streams to be processed is given in Fig. 2.

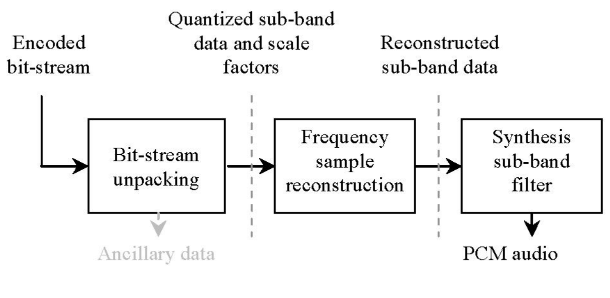
Encoded bit-stream
Quantized sub-band data and scale factors
Reconstructed sub-band data
Bit-stream unpacking
Frequency sample reconstruction
Synthesis sub-band filter
Ancillary data
PCM audio

Figure 1. DAB audio frame decoding

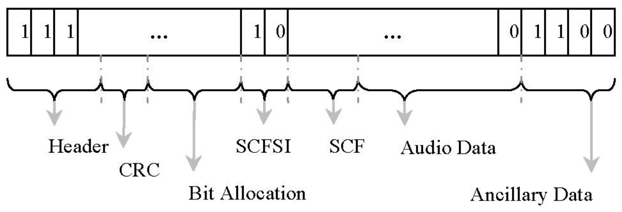

Figure 2. A DAB frame

A. Bit-Stream Unpacking

The purpose of the bit-stream unpacking block is to synchronize to the incoming bit-stream and extract frame related data necessary for the reconstruction of the samples, such as the scale factors and the sample word lengths, as well as the ancillary data packed by the service provider. The block diagram of the bit-stream unpacking block is given in Fig. 3.

1) The Controller

The controller is responsible for maintaining proper operation of the decoding process. It is supported by three sub-blocks, Synchronize & Extract Header, CRC check and the decode mode pointer.

The synchronization of the incoming bit-stream is handled by the Synchronize & Extract Header sub-block, which synchronizes and extracts the 32 bit header information. The header contains a synchronization word followed by frame related data such as the bit rate, sampling frequency and frame mode. As DAB uses a constrained version of the MPEG Audio Layer II – LFE coding algorithm, some of the frame related data bits are known a priori, such as the layer and sampling frequency bits. These bits are also included in the synchronization process to reduce the chance of misdetection, as the synchronization word itself is not a forbidden word, and can be found elsewhere in a frame.

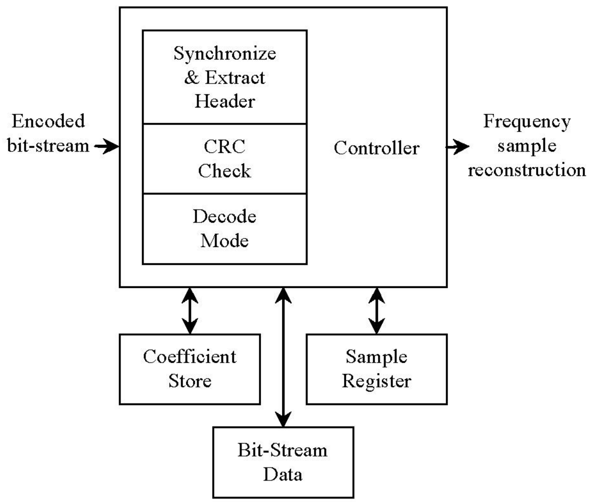

Figure 3. Block diagram of the bit-stream unpacking block

Upon completing the synchronization and extracting the header, the CRC-16 code word that follows the header is written to the CRC block to be checked later on. A DAB audio frame uses an additional CRC-8 word to protect the scale factors, in addition to the CRC-16 word that the MPEG Audio Layer II – LFE coding algorithm utilizes to protect the last 16 bits of the header along with the bit allocation and scale factor select information parts of the frame. As the bit-stream processor was designed to operate bit-serially, the CRC check was implemented as given in [2].

The final sub-block used by the controller is the decode mode pointer, which is used to determine which section of the frame is currently being processed and what decoding steps will follow.

2) The Coefficient Store

The coefficient store contains standard related coefficients, such as the re-quantization constants and bit-allocation tables that will be used later on for frequency sample reconstruction purposes. However, not all coefficients are stored as they appear in [1], due to the efficient implementation of the frequency sample reconstruction block, which will be explained in Section II B.

3) Bit-Stream Data

The bit-allocation, SCFSI and scale factor index information extracted from the frame is stored in the bit-stream data block. The bit allocation data is stored in a table that is capable of scanning through the sub-bands continuously, thus simplifying the audio sample extraction process. The SCFSI and scale factor indexes share storage to reduce the overall size of the bit-stream processor, as they are not used simultaneously. Keeping the addressing of the storage elements based on non-zero bit allocated sub-bands eliminates the need to check each sub-band for bit allocation information each time the data needs to be accessed. The controller provides the bit-stream data block with the sub-band number, and receives the bit allocation and the SCFSI or scale factor index, depending on which stage of the decoding is in process.

4) Sample Register

The sample register is used to temporarily store the audio sample until it is completed, as these samples are read bit-serially. Depending on whether these bits correspond to a sample or a grouped sample code word, the ordering and meaning of the bits also change, which is handled by the sample register.

B. Frequency Sample Reconstruction

The audio samples are decoded and reconstructed using the information obtained from the header, bit allocation, scale factor select information and scale factor indexes. However, due to the complex encoding algorithm, the reconstruction of the samples requires a moderate amount of operations. The frequency sample reconstruction block handles these operations in a computationally efficient manner to minimize the area and reduce the power consumption of the device. The necessary computations for the reconstruction process of each sample can be seen in (1).

$$PCM = SCF \times C \times (Sample + D) \quad (1)$$

where SCF is the scale factor extracted from the frame, and C and D are constants determined from the bit allocation information and are given in [1] and [2]. However, the samples are not always directly coded in the bit-stream. Depending on the results of the psychoacoustic analysis, the encoder may pack three samples into one sample code word that will have to be decoded in order to obtain the samples. The packing is done in accordance with (2).

$$v_i = i^2 \cdot x + i \cdot y + z \quad (2)$$

where i = 3, 5, or 9 and x, y and z are the individual samples. The de-grouping algorithm requires a modulus operation followed by an integer division operation executed three times to obtain three samples from one code word. The first modulus operation provides the sample z, while the integer division operation provides the input for the next cycle of the modulus operation. The second and third modulus operations will provide y and x, respectively.

A simplified block diagram of the direct implementation of the frequency sample reconstruction block is given in Fig. 4. It can be seen from the block diagram that the implementation requires an adder, two multipliers and a divider. However, dividers aren't very efficient when considered for their power consumption or size [4].

An alternative structure capable of handling these computations is given in Fig. 5. The proposed structure contains no dividers, due to the fixed number of divisors needed to evaluate (2).

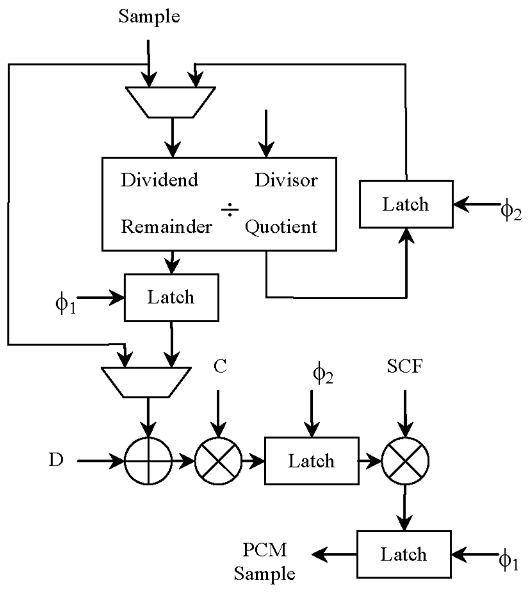

Figure 4. Direct implementation block diagram

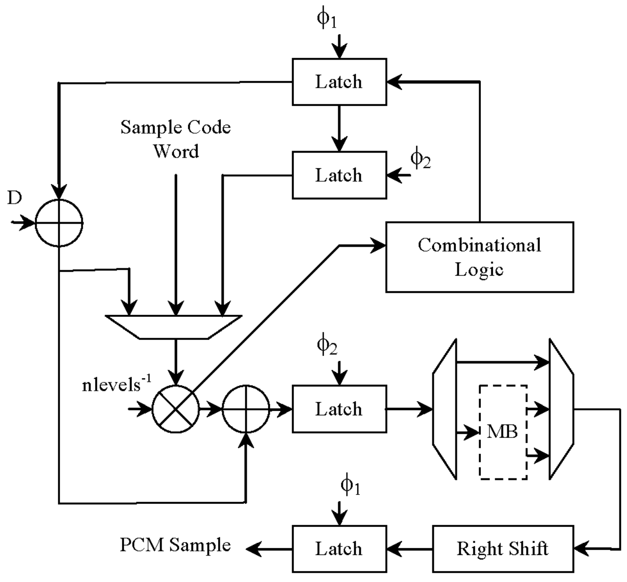

Figure 5. The ALU block diagram

The division and modulus operations are only ever needed when three samples are grouped into one sample code word and a de-grouping operation is necessary. In this case the decoded sample code word will be divided into 3, 5 or 9, depending on the number of quantization levels obtained from the bit allocation data. Exploiting this, the division operation is carried out as a multiplication with the inverse of the divisor, i.e. nlevels^{-1}.

The integer part of the result of this multiplication will give the required integer division result. Since *i* can only take on three values, the fractional part of the result of the multiplication can be mapped to the 17 possible values for the remainder of the division operation through a simple combinational logic block.

Closer inspection of the coefficient C reveals that this coefficient is equivalent to nlevels^{-1} + 1. Thus, multiplication by C is also handled by the multiplier responsible for division, with an extra adder at the output of the multiplier to compensate for the + 1 factor. The coefficient D is extracted from the bit allocation index value. This reduces necessary storage, as there is no need to store the coefficients C and D anymore.

The final reduction in computational efficiency is introduced in the rescaling part of the reconstruction of the audio samples. The 64 scale factors necessary for rescaling the audio samples can actually be obtained from three base numbers and their division by the powers of 2, thus eliminating the need to store all 64 scale factors in the coefficient store. As one of these three base values is actually 1, the selection of any scale factor produced from this base will result in a division by the required power of 2, which is merely a shift operation without any multiplication involved. Exploiting this property of the scale factors, one can implement the multiplication by the scale factors as a multiplier-block [5], as seen in Fig. 6. The multiplier block consists of 4 adders and 4 hardwired shift operations. The first adder to the left is fed with the input (x1) and its 2-bit shifted version (x4), to produce 5 times the input (x5). Similarly, the second one is fed with the output of the first adder and its 4-bit shift to produce 85 times the input, the third with the output of the second adder and an 8-bit shift to produce 21845 times the input, and the final one with the input and a 1-bit shift of the output of the third adder to produce 43691 times the input. The de-multiplexer in Fig. 5 is used to bypass the multiplier block when necessary. Having the scale factor values inherent in the multiplier block architecture renders the storage of any of the scale factors unnecessary. Thus, effectively the 64 scale factor storage block along with an NxN combinational multiplier, where N is the number of bits, is replaced by 4 adder stages, 4 hardwire shifts and a variable shifter.

A comparison of the resources required by the two implementations is made in Table 1. In order to make the comparison meaningful the direct implementation was also made to be as efficient as possible, implementing the combinational divider as an efficient N/N integer divider [6], with its size equivalent to $N^2/2$ adder cells, and keeping the controller logic unchanged where possible. The operations that take place in each clock cycle for the worst case of having to de-group three samples from a three bit sample code word for the direct implementation and the proposed implementation are listed in Table 2. The last two clock cycles need to repeat twice more to complete the de-grouping process.

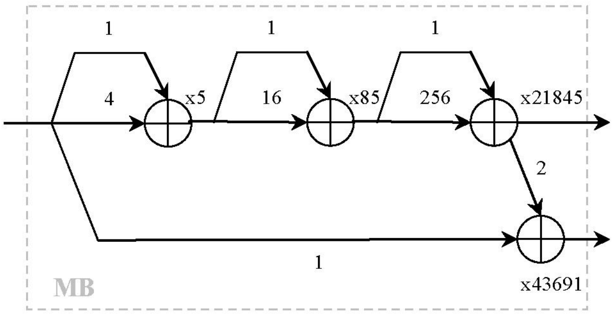

Figure 6. Scale factor multiplier block

TABLE I. RESOURCE COMPARISON

Resource Type	Resource Size	
	Direct Implementation	*Proposed Implementation*
Memory	• Bit Allocation Index Storage • Scale Factor Selection Index / Scale Factor Storage	
Coefficient Stores	• nlevels (16xN) • D (16xN) • C (16xN) • Scale Factors (64xN)	• nlevels^{-1} (16xN)
Arithmetic Blocks	• NxN multiplier (x2) • N-bit adder (x1) • N/N divider (x1)	• NxN multiplier (x1) • N-bit adder (~x6)

TABLE II. OPERATIONAL COMPARISON

Clock Cycle	Executed Operations	
	Direct Implementation	*Proposed Implementation*
ϕ_1 Initialize	• N/N comb. division • Store remainder in latch	• NxN comb. multiplication • Store combinational logic output in latch
ϕ_2 Loop (x 3)	• Add D to the stored remainder • Multiply by C • Store result in latch • Store quotient in latch	• Add D to stored sample • Multiply by nlevels^{-1} • Add results • Store in latch
ϕ_1 Loop (x 3)	• Multiply result by SCF • Store result in output latch • N/N combinational division • Store remainder in latch	• Multiply by SCF • Shift as necessary • Store result in output latch • NxN combinational multiplication • Store combinational logic output in latch

III. FUTURE WORK

The proposed frequency sample reconstruction block contains one more multiplier. However, closer inspection reveals possibilities for further reductions in computational complexity, size and power. The aforementioned multiplier is responsible for handling the multiplication with nlevels^{-1}, which has only 16 different values. Thus, this multiplier could be implemented as a reconfigurable multiplier block (ReMB) for multiple constant multiplications as proposed in [3]. The proposed work will be carried out until the submission date for the final paper.

IV. CONCLUSIONS

In this paper the design of a computationally efficient bit-stream processor for MPEG Audio Layer II LFE targeting DAB audio receivers was described. Manipulation of the inherent constants revealed a simpler structure for handling the re-quantization and re-scaling of the audio samples, which resulted in a reduced size coefficient store as well as a computationally efficient frequency sample reconstruction block implementation.

REFERENCES

[1] ETSI, "Radio broadcasting systems; Digital audio broadcasting (DAB) to mobile, portable and fixed receivers", ETS 300 401, 1997.

[2] ISO, "Information technology – Generic coding of moving pictures and associated audio information – Part 3: Audio", ISO/IEC 13818-3, 1998.

[3] S.S. Demirsoy, A.G. Dempster and I. Kale, "Design guidelines for reconfigurable multiplier blocks", IEEE International Symposium on Circuits and Systems, 2003, vol.4, pp.289-292, May 2003

[4] S.F. Oberman and M.J. Flynn, "Division algorithms and implementations", IEEE Transactions on Computers, vol.46, no.8, pp.833-854, August 1997

[5] A.G. Dempster and M.D. Macleod, "Use of minimum-adder multiplier-blocks in FIR digital filters", IEEE Trans. CAS-II, vol.42, no.9, pp.569-577, November 1995

[6] K.Y. Khoo and A.N. Willson, Jr., "Efficient VLSI implementation of n/n integer division", IEEE International Symposium on Circuirs and Systems, 2005, vol.1, pp.672-675, May 2005