Mp3 Decoder Design & Implementation Using VHDL

Ravindar Sharma[1] Ghanshyam Kumar Singh[2] Ram Mohan Mehra[3]
[1,2,3]Department of ECE,School of Engineering & Technology
[1,2,3]Sharda University, Greater Noida, UP, India.

Abstract— MP3 has become one of the most popular standards for digital audio and video broadcasting. High compression ratios offered by MP3 codecs in various standalone players and hand held devices over the last few years has increased its popularity immensely. Internet users, music lovers who would like to download highly compressed digital audio files at near CD quality are the most benefited. Psychoacoustic model, Modified Discrete Cosine Transform (MDCT) and Huffman coding play a vital role in achieving such magnificent compression ratios. In this work, the performance and result of each different block of Mp3 decoder is shown in VHDL using Modalism as the simulation tool. Testing and simulation were made to ensure full functionality of the design. An MP3 song of 128bitrate, 44 KHz and single channel is selected for the simulation. The simulation results show the effectiveness to implementation of Mp3 decoder in VHDL.
Keywords: MP3, Decoder, VHDL

I. INTRODUCTION

The mobile multimedia service has changed people's daily lifestyles. It is not difficult to find people who listen to music or watch TV via their mobile devices such as mobile phones, MP3 players, and portable media players. People can have a variety of experiences with their mobile devices. MPEG Layer-3, otherwise known as MP3, has generated a phenomenal interest among Internet users, or at least among those who want to download highly-compressed digital audio files at near-CD quality [1-3]. In this work, the decoding process of MP3 decoder is presented. Incoming MP3 streams are fed into the input module of the decoder and processed. If the synchronization word is found and CRC word is successfully checked, the output main data containing scale factors and Huffman code bits are sent to a predefined buffer. Huffman decoder reads data from buffer and decodes them. The decoded scale factors are delivered to requantizer component. The output 576 frequency lines are written to main memory. Using the input scale factors, requantizer descales the 576 small integer numbers output from Huffman decoder. Reordering process is applied to reorder the frequency lines including short window blocks. The functionality of the last four components--- anti alias, IMDCT, frequency inversion and filter bank is to transform the MP3 from frequency domain to time domain. By merging frequencies using butterfly calculations, anti-alias reduces the alias effects introduced from the non-ideal Bandpass filter used in encoding process. In IMDCT, 576 frequency lines are divided into 32 sub bands with 18 frequency lines in each. Applying multiplications and cosine calculations to each 18 samples depending on four classes of window type, IMDCT generates samples for filter bank component. After the frequencies of the input samples are inverted by frequency inversion, filter bank exploits MCT to translate the aliased signals and filter out the undesired aliasing in translated signals by using windowing. Hence, the signals in frequency domain are converted back to their time domain origins.

A. II. DECODER DESIGN FLOW CHART

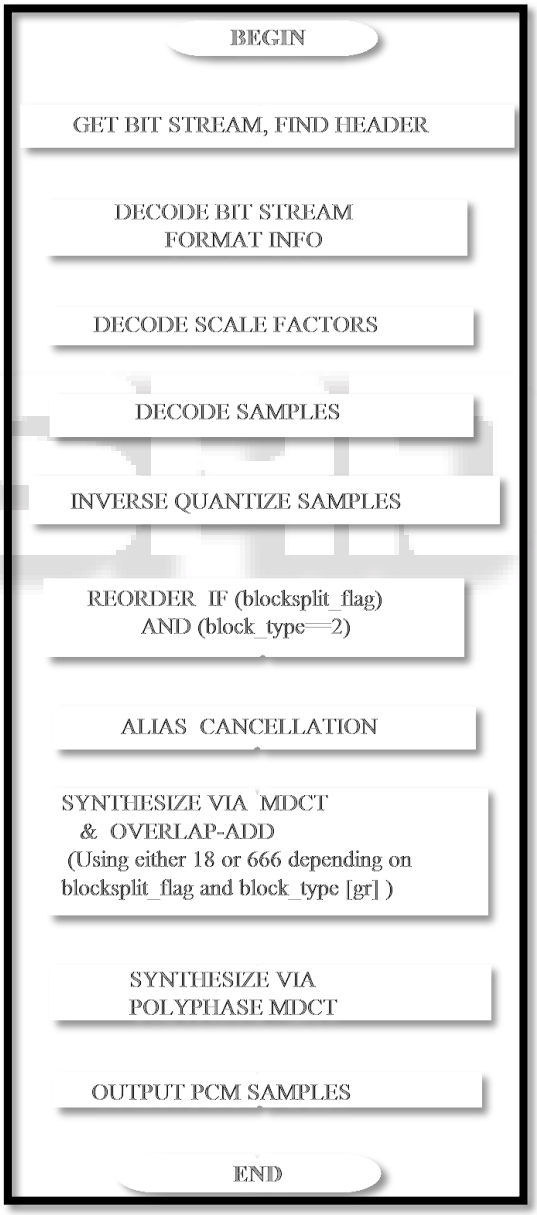

Fig. 1: MPEG 1 Layer III decoder flow chart

II. MP3 DECODER OPERATION

The quantized samples are derived from the Huffman codes in the Huffman decoding block. The necessary side information needed for Huffman decoding is obtained from

Huffman Info decoding block. Since the Huffman codes are variable length codes, the Huffman encoding of the quantized samples results in a variable frame size. In order to optimize the space usage in a frame, the data from the adjacent frames are packed together. So, the *Huffman Decoding* stage refers to the previous frames data for its decoding. The next step after Huffman decoding, is the re-quantization.

The re-quantizer re-quantizes the Huffman decoder output using the scale factors and the global gain factors. The re-quantized data is reordered for the scale factor bands. The re-quantized output is fed to the stereo decoder, which supports both MS stereo as well as Intensity stereo formats. The alias reduction block is used to reduce the unavoidable aliasing effects of the encoding poly phase filter bank. The IMDCT block converts the frequency domain samples to frequency sub band samples. The frequency sub bands were introduced by the encoder.

A. Huffman decoding:

The MP3 quantized samples are encoded using the variable length encoded samples. In Huffman encoding the Huffman codes are produces from input symbols using loss-less coding scheme. The Huffman codes are mapped based on the statistic contents of the input sequence. There are 32 Huffman tables that are used to map the Huffman code bits with the symbols x and y. The Huffman decoder compares the input Huffman code bits with the 32 Huffman code tables to find a match to represent the Huffman code bits. The Huffman code tables are predefined base on statistics of ISO standard 11172-3. The values of table select in the side information provides the information on which standard table is used.

B. Inverse Modified Discrete Cosine Transform

The Inverse Modified Discrete Cosine Transform (IMDCT) transforms each subband from frequency domain to time domain. It in cooperated with the synthesis polybasic filter bank to produce the time samples *xi* from the input frequency line X(k). The IMDCT calculation is obtained by using equations 1, where $0 \le i \le n-1$.

$$xi = \sum_{k=0}^{\frac{n}{2}-1} X_k \cos\left[\frac{\pi}{2n}\left[2i+1+\frac{n}{2}\right](2k+1)\right] \text{----------(1)}$$

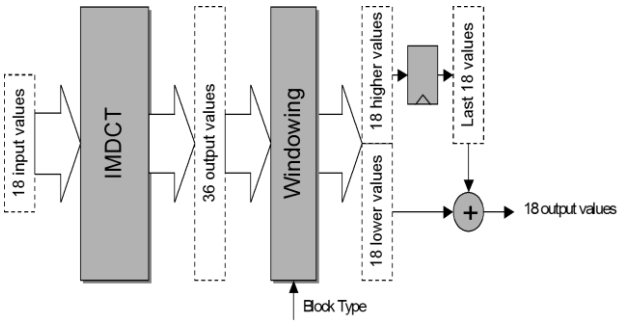

Fig. 2: IMDCT design flow

Where n is the number of the windowed samples, n is equal to 12 for a short block and n is equal to 36 for a long block. For a case with n = 36, the IMDCT is an 18 point DCT that generate 36 poly phase filter sub-band samples from 18 input frequency lines. These samples are multiplied with a 36 point window before it can be used by the next step in the decoding process.

1) *C. Frequency inversion*

The overlap output consists of 18 time samples for each 32 poly phase sub bands. All the odd subsamples in the odd sub bands are negated by multiplying by -1 before processing the time samples into synthesis poly phase filter bank.

III. RESULTS DISCUSSION

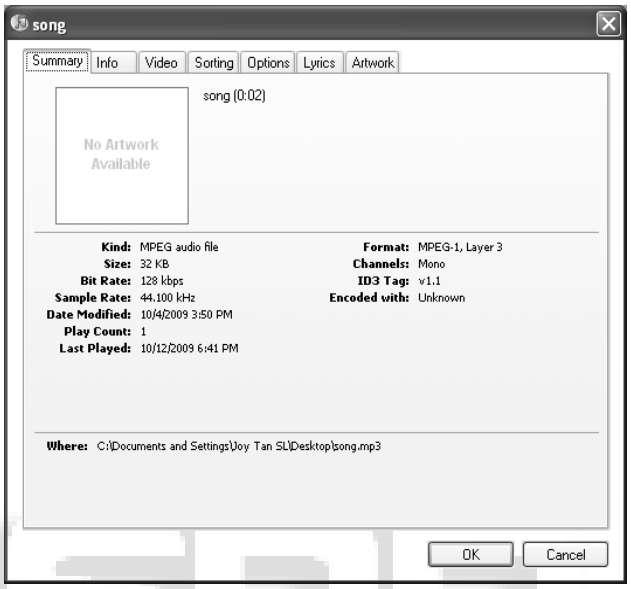

Fig. 3: MP3 song detail

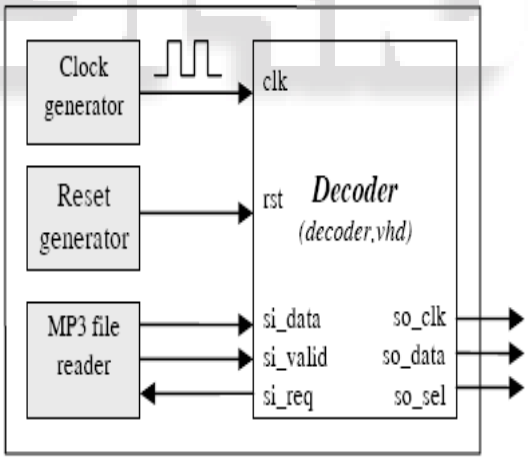

Fig. 4: The MP3 decoder test-bench

The performance and result of each different block implemented in VHDL using Model Sim as the simulation tool is presented. Testing and simulation were made to ensure full functionality of the design. An MP3 song of 128bitrate, 44 KHz and single channel is selected for the simulation, Figure 3 shown the details of the MP3 song selected using MP3 player.

The VHDL source code cannot read directly from MP3 files therefore a C+ code is written to convert the MP3 song to a bitstream (bit by bit binary file). The code written is found under appendix A. The converted binary file is to be used as the input file for the MP3 decoder test bench. The MP3 test bench architecture is shown in Figure 4. The MP3 decoder

declarations and test bench source code can be found in appendix A.3 and appendix A.11 respectively.

A. A.CONTROLLER

The controller block is designed to control the state of the 10 blocks of the MP3 decoder. The waveform in Figure 5 shows the output of the controller block at the synchronizer block.

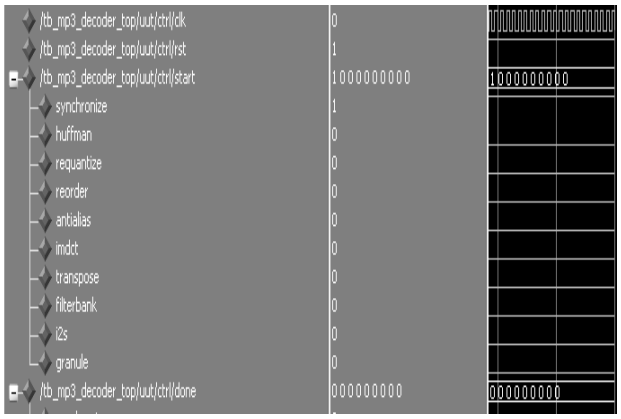

Fig. 5: Waveform output of controller block at synchronizer block

B. .SYNCHRONIZER

The synchronizer block is the first stage in the pipeline. It interfaces with the input MP3 bitstream. It will verify for the correct MP3 file and find the beginning of a new frame. When a correct MP3 file is read it will read the header and side information. The waveform in Figure 6 shows the header data and Figure 7 shows the side information data. Finally the main data is read and transferred to the bit reservoir (bit reservoir.vhd) the waveform result is shown in Figure 8.

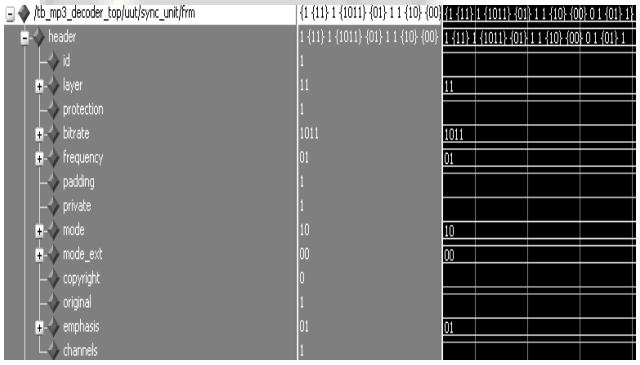

Fig. 6: Waveform output of header from synchronizer

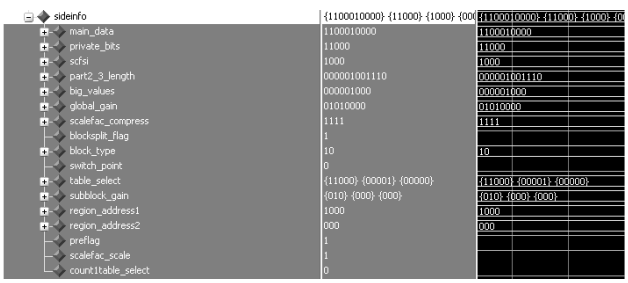

Fig. 7: Waveform output of side information from synchronizer

The frame information will be needed by all blocks using the frame input signal. frame.header. frequency is written to extract the frequency of the frame.

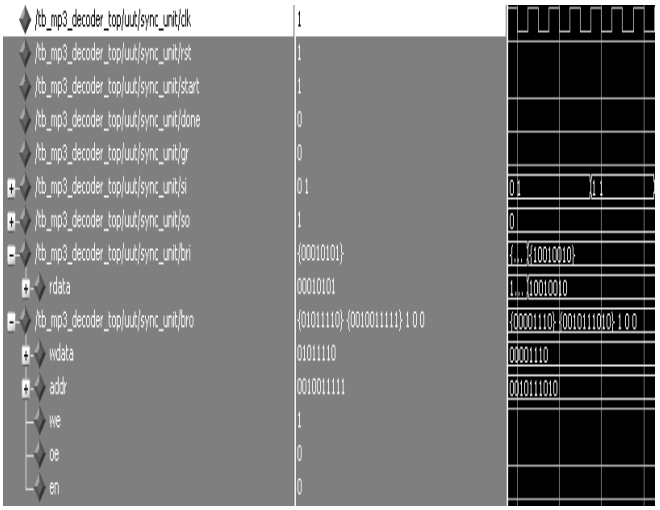

Fig. 8: Waveform output of main data read and transfer to bit reservoir

C. HUFFMAN DECODER

The Huffman coding is a loss-less source coding scheme which uses the statistical information from the input sequence to produce the output sequence. That is a prefix free code for easy decoding. Which means an input sequence with high probability is coded with a short codeword. The 32 Huffman tables based on statistics for audio information are referenced from working C code. The ROM created stores the 32 Huffman code tables as shown in Figure 9. The side information provides the information on which Huffman table is to be used for current frame therefore the frame data are read into the Huffman block as shown in Figure 10.

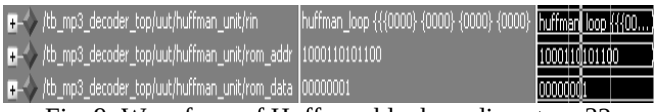

Fig. 9: Waveform of Huffman block reading store 32 Huffman table from ROM.

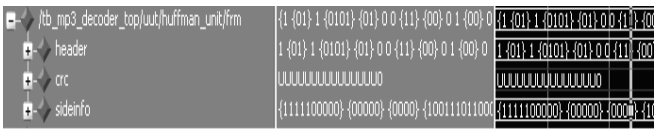

Fig. 10: Waveform of Huffman block reading frame data from synchronizer

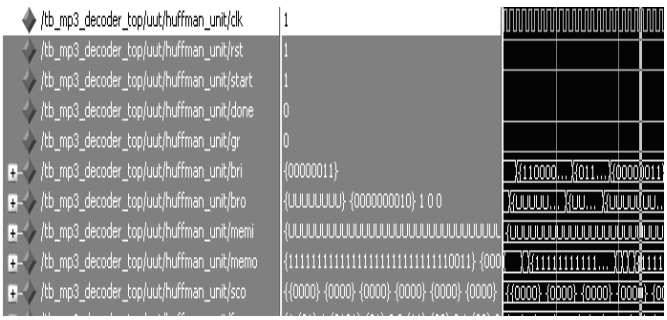

Fig. 11: Waveform of Huffman blocks reading input and output from bit reservoirs and main memory.

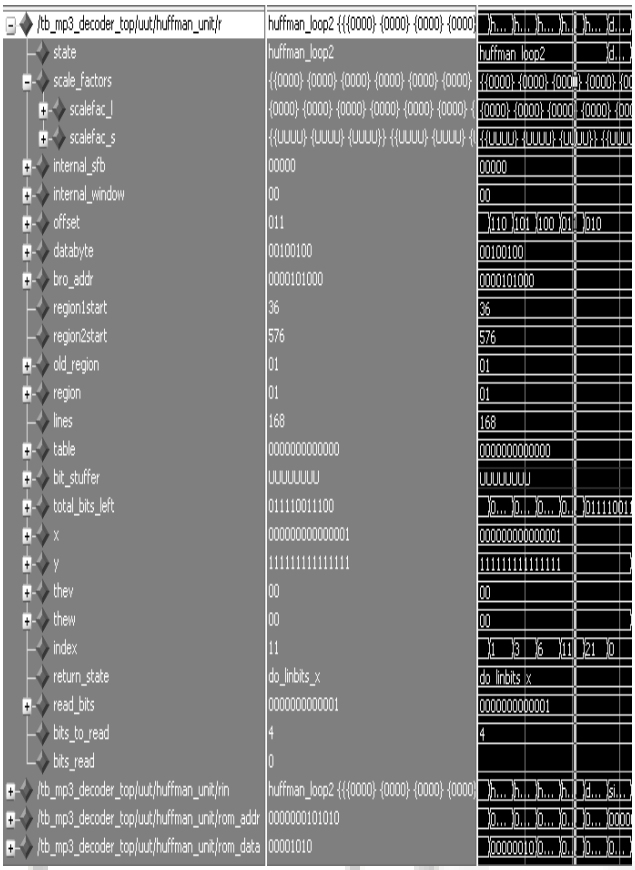

Fig. 12: Waveform of Huffman blocks output at Huffman loop2 state.

The Huffman block is to transform the input data into scale factors (sco) and symbols to represent the 576 frequency lines. It receives the incoming data from the bit reservoir (bri/bro) and places the result in the main memory (memi/memo) as shown in Figure 11

The Huffman block starts by reading and decoding the scale factors, information that are required by the requantizer block. The second part of Huffman block decodes on the rest of the information until all the Huffman code bits have been decoded or until all the 576 frequency lines have been decoded.

D. REQUANTIZER

The task of the requantizer is to rescale the Huffman decoded scaled and quantized frequency lines by using the scale factors decoded from Huffman block.

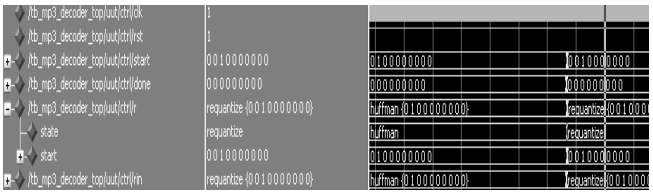

Fig. 13: Waveform of controller block at requantizer

The global scaling information is applied to all frequency lines. From the result the block type 0 is read at calculation (calc*)* state the frequency lines for long block is calculated. It adjusts critical band boundary with value *next_sfb_boundary* is calculated that depends on the values of sfb and freq value as shown in Figure 14. It also shows the calculated gain_index and gain_exp which read in values from the side information.

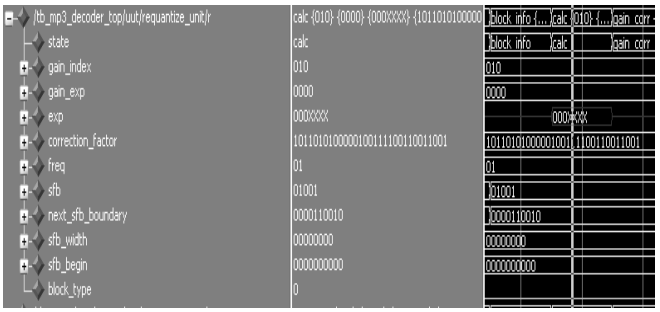

Fig. 14: Waveform of output for requantizer block

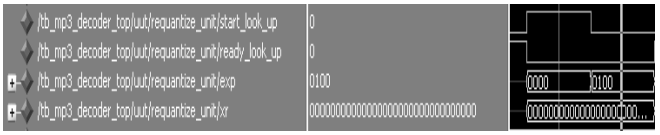

Fig. 15: Waveform of look_up table of requantizer block

Fig. 16: Waveform of gain_correction table of requantizer block.

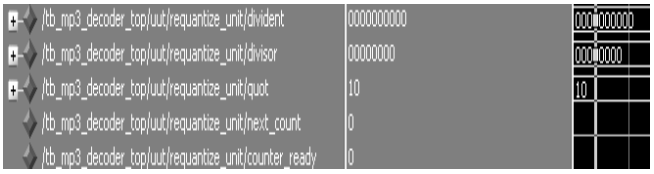

Fig. 17: Waveform of divider_win block of requantizer block.

The design implemented consists of a table_look_up to perform the look-up as shown in Figure 15. The simulation shows when the ready_look_up is set and start_lookup is not set no data is being process. When the start_lookup is being set data is processed. The gain_correction is another table that stored correction factor as shown in Figure 16.

The divider_win is created for the window calculation shown in Figure 17 the values are used in the cal state of requantizer. The shifter is used in gain_corr and shift state for multiplying. The output of shifter data are shown in Figure 18.

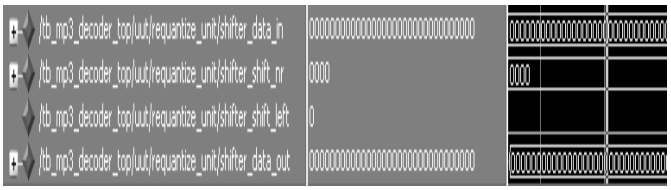

Fig. 18: Waveform of shifter block of requantizer block.

E. REORDER

The task of the reorder is to sort the 576 frequency lines that are created by the requantizer block. As discussion the reordering will only be applied to short blocks (block type =10) as only short blocks have more than one window. Referring to section 4.1.3 for a block type 11 indicated it is an End block as shown in Figure 19. Therefore reorder is not required for the MP3 song selected to run this simulation.

Fig. 19: Waveform of Block_type of side information read from requantizer.

F. ANTIALIAS

The task of antialiasing is to implement the butterfly calculation. The butterfly reads the input data from memory design with some counters and multiplier.

Fig. 20: Waveform of antialias butterfly calculations

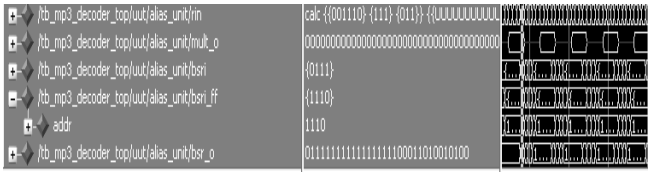

Fig 21: Waveform of calculated 8 pairs of alias in RAM

There are 32 rows with 8 butterflies and each butterfly has 4 multiplications and 2 additions. The calculation data of row 14 and butterfly 8 are shown in Figure 20. The figure also shows the data being read to and from address line x1 and x2. The calculated 8 pairs of alias are placed in RAM as shown in Figure 21.

IV. CONCLUSION

The decoding process of MP3 decoder is presented. If the synchronization word is found and CRC word is successfully checked, the output main data containing scale factors and Huffman code bits are sent to a predefined buffer. Huffman decoder reads data from buffer and decodes them. The decoded scale factors are delivered to requantizer component. The output 576 frequency lines are written to main memory. Using the input scale factors, requantizer descales the 576 small integer numbers output from Huffman decoder. Reordering process is applied to reorder the frequency lines including short window blocks. The functionality of the last four components--- anti alias, IMDCT, frequency inversion and filter bank is to transform the MP3 from frequency domain to time domain. By merging frequencies using butterfly calculations, anti alias reduces the alias effects introduced from the non-ideal Bandpass filter used in encoding process. In IMDCT, 576 frequency lines are divided into 32 sub bands with 18 frequency lines in each. Applying multiplications and cosine calculations to each 18 samples depending on four classes of window type, IMDCT generates samples for filter bank component. After the frequencies of the input samples are inverted by frequency inversion, filter bank exploits MCT to translate the aliased signals and filter out the undesired aliasing in translated signals by using windowing. Hence, the signals in frequency domain are converted back to their time domain origins. The design shows significant enhancement in Mp3 decoder operation.

It was not so long ago that the average pop song converted into a .wav file took hours to download on a 28.8 kbps modem connection and at up around 50 megabytes of disc space. With the same song converted into an MP3 file, download time gets reduced dramatically to around one-tenth the original size while sounding just as good as before. The blocks of the MP3 decoder were implemented in VHDL and it would be more interesting to test it on a physical FPGA development board.

REFERENCES

[1] Niazi, M.F.; Seceleanu, T.; Tenhunen, H., "An automated control code generation approach for the SegBus platform," SOC Conference (SOCC), 2010 IEEE International , vol., no., pp.199,204, 27-29 Sept. 2010

[2] Hoang-Anh Pham; Van-Hieu Bui; Anh-Vu Dinh-Duc, "An Adaptive Huffman Decoding Algorithm for MP3 Decoder," Electronic Design, Test and Application, 2010. DELTA '10. Fifth IEEE International Symposium on , vol., no., pp.153,157, 13-15 Jan. 2010

[3] ISO / IEC 11172-3: Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1.5 M bit/s – Part3: Audio, ISO/ IEC 1993.

[4] M. Ruckert, Understanding MP3, Viewer, 2005, ISBN 3-528-05905-2.

[5] Bradenburg K. and Popp H., "An introduction to MPEG Layer-3", EBU Technical review, June 2000.

[6] Bradenburg K., "MP3 and AAC explained", in Proc. of the AES 17th Int. Conf. on high quality audio coding, 1999.

[7] Raissi R., "Theory behind MP3", 5th October 2010,

[8] Mathew M., Bhat V., Thomas S.M., Yim C., "Modified MP3 Encoder using complex modified discrete cosine transform" , ICME 2003.

[9] Fältmann I, Hast M, Lundgren A, Malki S, Montnemery E, Rångevall A, Sandvall J,Stamenkovic M., "A Hardware implementation of an MP3 decoder" , Digital IC project, LTH, Sweden , May 2003.

[10] Geoff Nicholson, "MP3 Explained: A Beginners guide" 18th January 2006, <http://www.hitsquad.com/smm/news/9903_109/?nl9905

[11] S. Haker, MP3: The Definitive guide, O'Reilly, 2000, ISBN 1-56592-66 1-7.

[12] Predrag Supurovic, "MPEG script", 10th January 2006, <http://www.dv.co.yu/mpgscript/mpeghdr.htm>.

[13] Gabriel Bouvigne, "MP3 Tech", 5th October 2005, <http://www.mp3-tech.org>.

[14] "MP3converter ", 15th December 2005, <http ://www.mp3-converter.com>.

[15] "Id3 tags", 28th December 2005, <http://www.id3.org

