A High-payload Audio Watermarking Scheme for Real time Applications

Ramin Almasi and Hooman Nikmehr

Digital audio watermarking is proposed as a tool to detect the digital audio data possession. Here, a method is presented which can be adopted in real time applications due to its high speed at embedding and detecting process. This method has higher payload in comparison with similar methods. In this algorithm, a time-domain synchronization is applied to find the exact position of the watermark in audio signals. To embed the watermark, first the audio signal is split into blocks. Then the watermark is embedded into the FFT components with greater magnitudes of audio signal. Unlike common audio watermarking methods that use the 1-bit embedding approach, here, the 2-bit embedding approach is presented which doubles the embedding payload compared to similar works. The experiment results on different signals demonstrate that this algorithm is faster and more transparent compared to similar algorithms. It is robust against common attacks like additive noise, MP3 compression, low pass filtering, re-sampling and re-quantization. The time complexity of proposed algorithm is compared with similar algorithms. Hardware simulation of synchronization code detection is conducted by FPGA.

Keywords: Payload, real-time audio watermarking, robustness, signal processing attacks

I. Introduction

Nowadays, broadband Internet connections and nearly error-free transmission of data facilitate the distributing multimedia files. Since 1992, number of published articles on digital watermarking has drastically increased. Although these algorithms primarily are developed for still images and video streams, interest and research on audio watermarking commenced later. In a sense, audio watermarking methods are categorized in two; time and frequency domains. In these algorithms the main issues are transparency, payload, reliability and robustness. In [1] a robust and high quality audio watermarking method is proposed in time domain applies amplitude modification in low frequencies. It divides the audio signal into group of samples, by using differential average of absolute amplitude in a group of samples for embedding the watermark into the host signal. This method defines thresholds frequently based on HAS [1] and audio signal; therefore increases computing and time complexity of the algorithm. This algorithm is not suitable for real-time applications. Researchers in [2] proposed a real-time audio watermarking technique using wavetables. In their method the PCM[2] wave forms are held in wavetables. Results of experiments and several attacks on different signals indicate that this is faster and more transparent when compared to other algorithms. In addition, it is robust against attacks such as additive noise, MP3 compression and low-pass filtering. It is obvious that this method is suitable for digital music instruments. In [3] a blind

Ramin Alamsi (phone: *0098 9355912762*, email: almasi.ramin@gmail.com) , *Hooman Nikmehr* (email: nikmehr@eng.ui.ac.ir) are with *Facultyof Computer Engineering, University of Isfahan, Isfahan, Iran.*

[1] Human Auditory System

[2] Pulse Code Modulation

audio watermarking method is proposed using self-synchronization, where the synchronization bits are embedded into time domain by modifying the audio signal time samples and watermark bits are embedded into DCT[3] transform components. This is not enough robust against the attacks. The method presented in [4] embeds the synchronization and watermark bits into low frequency components of the wavelet transform. Although this method leads to more robustness against attacks, the SNR[4] highly decreases and the audible noises are produced. The authors of [5] propose a fast audio watermarking scheme where the synchronization and the watermark bits are embedded into the time domain and the FFT domain, respectively. This method is robust against the attacks and signal processing operations and because of its time domain synchronization embedding process, it is suitable for real-time applications; however it has the payload of 30 bps that is not enough in comparison with similar audio watermarking methods. In [6], the synchronization bits are embedded into the low frequency components of DWT[5] transform. To embed the watermark bits, first the DWT transform is applied to the audio signal and then the DCT transform is applied to the low frequency components resulted from DWT transform. Next, the watermark bits are embedded into the DCT components. Since synchronization bits are embedded into the DWT domain, this algorithm is robust enough against the common attacks, while it takes more time to detect the synchronization bits and it cannot be suitable for real-time applications.

In this article a time domain synchronization algorithm is adopted suitable for real-time applications. To embed the watermarks, the audio signal is split into several blocks; Next the watermark bits are embedded into FFT components of greater magnitude, in a manner that the payload of embedded watermark into the host signal becomes double compared to similar works without significant changes in watermarking evaluation results such as SNR, ODG[6] and BER[7].

II. Watermarking Algorithm

This algorithm is based on the presented algorithm in [5]. Since the amount of data in audio signal is high, if the watermark bits embed into entire audio signal, a great volume of the main memory would be occupied; therefore the audio signal is split into blocks and then the watermark bits are embedded into these blocks. The synchronization bits are embedded at the beginning of watermark bits to detect the position of the watermarks (Figure 1). In this proposed algorithm the synchronization bits are embedded into the time domain and watermark bits are embedded into FFT domain (Figure 2).

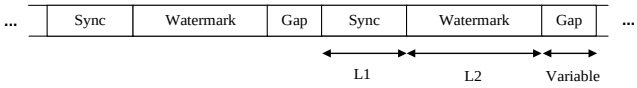

Fig. 1. Embedding segments.

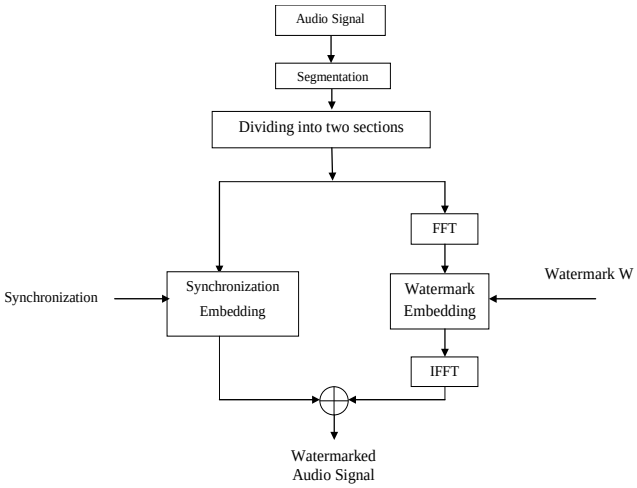

Fig. 2. Embedding scheme.

III. Synchronization Bits Embedding

In this article, the audio samples are considered to be normalized in [-1, 1] interval. If each sample is represented in *n* bits, the mentioned signal will be normalized in [-1, 1] interval by dividing each sample by 2^{n-1}. As illustrated in Figure 1, the synchronization code length is the *L1* samples. If n_{syn} samples are used for embedding each synchronization bit, in order to embed l bits of synchronization $L_1 = l.n_{syn}$ samples are required. The synchronization bits embedding algorithm is as follows: To embed synchronization bit "1" into n_{syn} samples of host signal, the inner samples are increased in a manner where their average becomes greater than the average of the extreme points. In order to embed bit "0", the inner samples are

[3] Discrete Cosine Transform

[4] Signal-to-Noise Ratio

[5] Discrete Wavelet Transform

[6] Objective Difference Grade

[7] Bit Error Rate

decreased so that their average becomes smaller than that of the extreme points (Figure 3). It is assumed that the synchronization bits are intended to be embedded into time samples as $s_p, s_{p+1}, \ldots, s_{p+n_{syn}-1}$ of host audio signal. A few notational keys are provided here.

s sampled time domain audio signal

s_p p-th sample of the signal

$\overline{s_{\text{int}}}$ average of inner samples

$\overline{s_{ext}}$ average of extreme points

$\delta_{\min}$ the minimum distance from the average of the extreme points for bit-embedding

φ the distortion introduced with respect to the average of the extreme points

n_{syn} the number of consecutive samples to embed each bit

The following embedding method is applied.

Algorithm 1. Synchronization embedding.

$$\overline{s_{\text{int}}} = \frac{1}{n_{syn}-2}\sum_{j=p+1}^{p+n_{syn}-2} s_j$$

$$\overline{s_{ext}} = \frac{s_p + s_{p+n_{syn}-1}}{2}$$

For $j = p$ **to** $p + n_{syn} - 1$ **Do**

$s'_j = s_j$

EndFor

To embed a "1"

If $\overline{s_{\text{int}}} < \overline{s_{ext}} + \delta$ **then**

$d = \overline{s_{ext}} + \delta - \overline{s_{\text{int}}}$

For $j = p+1$ **to** $p + n_{syn} - 2$ **Do**

$s'_j = s_j + d$

EndFor

EndIf

To embed a "0"

If $\overline{s_{ext}} < \overline{s_{\text{int}}} + \delta$ **then**

$d = \overline{s_{\text{int}}} + \delta - \overline{s_{ext}}$

For $j = p+1$ **to** $p + n_{syn} - 2$ **Do**

$s'_j = s_j - d$

EndFor

EndIf

In this article, 16-bit barker code "1111100110101110" is used as the synchronization code. The synchronization embedding parameters are $\delta_{\min} = 10^{-3}, \varphi = 0.05$ and $n_{syn} = 4$ such as [5].

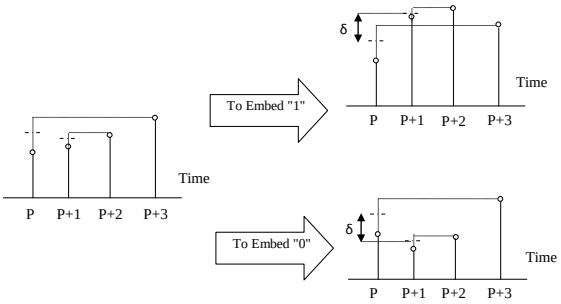

Fig. 3. Synchronization bit embedding.

IV. Proposed Algorithm for Watermark Embedding

Unlike common audio watermarking methods where 1-bit embedding approach is applied, here, the 2-bit embedding algorithm is applied that embeds the 2-bit string "00" , "01", "10" or "11" instead of bit "0" or "1".

The length of watermark is considered L_2 samples as shown in Figure 1. If each watermark bit is embedded in L_3 samples of audio signal, the watermark length (L_2) will be $|w|.L_3$ samples where $|w|$ is the watermark length in bits. The L_3 is considered as 512 [5]. The host audio signal is assumed as mono signal; nevertheless this algorithm is applicable on stereo signals. The watermark bits are embedded into FFT coefficients. In this method to embed each watermark bit, a block of L_3 samples of FFT coefficients is required. To obtain these, the FFT transform is applied on the block of

L_3 time samples. Here, s represents the time domain signal and S represents the FFT domain signal. After that suitable frequencies of S should be chosen for embedding the two watermark bits. Due to the symmetry property of the FFT, $S_{L_3-k} = S_k^{\ *}$ for $k = 0,1,2,...,\left(\frac{L_3}{2}\right)-1$, i.e. the second half of the sequence S_k in FFT coefficients, are the complex-conjugate values of the coefficients of the first half; hence the second half of the sequence is disregarded for embedding the watermark bits. This issue is considered in the highest index of (1) inequality. Also, the S_0 is not fit for embedding and is ignored. The series $\left\{S_1, S_2, ..., S_{\frac{L_3}{2}-1}\right\}$ is first sorted ascending in accordance with magnitude, such that S' satisfies

$$\left|\acute{S_1}\right| \le \left|\acute{S_2}\right| \le \left|\acute{S_3}\right| \le \cdots \le \left|S_{\frac{L_3}{2}-1}^{\acute{}}\right| \qquad (1)$$

Then the five consecutive elements of S' which are represented as: $S'_m, S'_{m+1}, S'_{m+2}, S'_{m+3}, S'_{m+4}$ $(1 < m < \frac{L_3}{2} - 4)$ are chosen to embed the two watermark bits. The m represents the position of the embedding watermarks in FFT coefficients. In this algorithm the embedding position is chosen in a manner where a balance is established between transparency and robustness. The larger m leads to less transparency and more robustness and vice versa. In this article $m = 250$ .Here, the M_k is applied to represent the magnitude of $\left|S'_k\right|$. Parameters A, B, C and D are introduced as follows:

$$\begin{aligned} A &= M_{m+4} - M_{m+3} \\ B &= M_{m+3} - M_{m+2} \\ C &= M_{m+2} - M_{m+1} \\ D &= M_{m+1} - M_m \end{aligned} \qquad (2)$$

At this stage, in order to embed bit strings "00", "01", "10" and "11" the following conditions are applied.

1. $\begin{cases} A \ge \alpha.B \\ C \ge \beta.D \end{cases}$ to embed "11"
2. $\begin{cases} A \ge \alpha.B \\ \beta.C \le D \end{cases}$ to embed "01"
3. $\begin{cases} \alpha.A \le B \\ \beta.C \le D \end{cases}$ to embed "00"
4. $\begin{cases} \alpha.A \le B \\ C \ge \beta.D \end{cases}$ to embed "10"

Where, α, β are considered to avoid the embedding overlaps of conditions. In case any of the corresponding conditions for embedding the two bits is not satisfied, the changes have to be made in the spectrum S' as follows:

1. To embed "11"

$$S'_{m+3} = \frac{M_{m+4} + \alpha.M_{m+2}}{(\alpha+1).M_{m+3}}.S'_{m+3}$$

$$S'_{m+1} = \frac{M_{m+2} + \beta.M_m}{(\beta+1).M_{m+1}}.S'_{m+1}$$

2. To embed "01"

$$S'_{m+3} = \frac{M_{m+4} + \alpha.M_{m+2}}{(\alpha+1).M_{m+3}}.S'_{m+3},$$

$$S'_{m+1} = \frac{\beta.M_{m+2} + M_m}{(\beta+1).M_{m+1}}.S'_{m+1}$$

3. To embed "00"

$$S'_{m+3} = \frac{\alpha.M_{m+4} + M_{m+2}}{(\alpha+1).M_{m+3}}.S'_{m+3}$$

$$S'_{m+1} = \frac{\beta.M_{m+2} + M_m}{(\beta+1).M_{m+1}}.S'_{m+1}$$

4. To embed "10"

$$S'_{m+3} = \frac{\alpha.M_{m+4} + M_{m+2}}{(\alpha+1).M_{m+3}}.S'_{m+3}$$

$$S'_{m+1} = \frac{M_{m+2} + \beta.M_m}{(\beta+1).M_{m+1}}.S'_{m+1}$$

In each case, these changes should be applied to the corresponding S_k and its complex-conjugate. It is easily demonstrated through mathematical operations that for the new S'_{m+1} and S'_{m+3} the following conditions are satisfied:

$$S'_m \le S'_{m+1} \le S'_{m+2} \le S'_{m+3} \le S'_{m+4} \qquad (3)$$

Where the previous sequence of series S'_k is maintained. As a result, in terms of reducing the computational load, this algorithm has significant advantages over [5].

V. Synchronization Detection

In this method, existence of a secret key guarantees the security of the method. The secret key includes information such as watermark length in bits, length of the synchronization part in bits, number of the required samples for each bit of watermark and synchronization, and positions of the sorted coefficients to embed the watermarks. In order to extract the bits of watermarks, first the synchronization bits are found in the watermarked signals, and then the bits of watermarks are extracted from L_2 of the next samples. When the length of watermark is variable, in the embedding process, “01111110” bit string is attached to the beginning and the end of watermark to boost the confidence of watermark detection and extraction operations.

For detecting the synchronization, its length in bits (L_1) and number of required samples per bit (n_{syn}) are required.

From the beginning of time domain signal, a sample window of $L_1 = l.n_{syn}$ is considered, the samples inside the window are divided into consecutive n_{syn} sets with no overlap. Therefore, for each set the 0 or 1 bit is detected according to the following algorithm:

Let $t_p, t_{p+1}, t_{p+2}, ..., t_{p+n_{syn}-1}$ represents the consecutive time domain samples of the watermarked signal, the synchronization detection algorithm is as follows:

Algorithm 2. Synchronization detection.

$$\overline{t_{int}} = \frac{1}{n_{syn}-2} \sum_{j=p+1}^{p+n_{syn}-2} t_j$$

$$\overline{t_{ext}} = \frac{t_p + t_{p+n_{syn}-1}}{2}$$

If $\overline{t_{\text{int}}} > \overline{t_{ext}}$ **Then** "1" detected
else "0" detected
EndIf

The above-mentioned algorithm indicates that if the inner points' averages are more than the average of the extreme points, the bit will be detected as "1", otherwise the bit will be detected as "0". As a result of this operation in this window, a $l-$ bit string is generated in this window, which is compared with the synchronization bit string considered in synchronization step. If these two bit strings are identical, they are synchronization bits; otherwise the above-mentioned operations will be repeated with one-sample shift until the end of the signal. Since the window generated by the n_{syn} shifts are common in 15 synchronization bits with previous window, the detection rate of synchronization bit string could be strongly accelerated through a recursive technique.

For this purpose n_{syn} sets are considered here and for each window shift from 1 to n_{syn}, the generated synchronization bit string is located in n_{syn} distinct variables. Afterwards, in the shifts’ numbers larger than n_{syn}, the suitable variable is selected according to remainder of dividing number of shifts by n_{syn}. Finally, the bit string will be shifted one bit to the left and the detected bit from samples numbered according to following algorithm will be attached as the least significant bit of string.

Numbering of samples used to detect attaching bit is as follows:

Number of window length + 1 + number of shifts - n_{syn}
Number of window length + 1 + number of shifts - $n_{syn}+1$
Number of window length + 1 + number of shifts - $n_{syn}+2$
.
. (4)

.
Number of window length + number of shifts

This process will be repeated for new variables. For instance, for $n_{syn} = 4$, window length = 64 samples and after 3 shifts bit string "1101010001010010" is stored as the synchronization bits. So, for 7 shifts, this variable is selected through n_{syn} variables and the most significant bit (first number of the above-mentioned string which is "1" here) is deleted. Considering the sample numbers 68, 69, 70, 71, the synchronization bit is extracted and will be inserted to right hand of the above-mentioned string as the least significant bit. For number of shifts 11 this procedure will be repeated.

VI. Watermark Extraction

After detecting the synchronization bit string, $L_2 = |w|.L_3$ numbers of the next samples of watermarked signal are considered for watermark extraction. For this purpose, first, the FFT is applied to these L_2 signal samples; next, the second half of the coefficients that were obtained from FFT is neglected in compliance with embedding process. Finally the coefficients are sorted in an increasing manner, and 2-bit string will be extracted according to algorithm (5).

Let M'_k for $k = 1,2,3,\ldots,\frac{L_3}{2} - 1$, these representatives for FFT coefficient magnitude. m represents the position of embedding watermarks in FFT coefficients mentioned in the secret key.

$$A' = M'_{m+4} - M'_{m+3}$$
$$B' = M'_{m+3} - M'_{m+2}$$
$$C' = M'_{m+2} - M'_{m+1}$$
$$D' = M'_{m+1} - M'_m \qquad (5)$$

The watermark extraction algorithm is as follows:

Algorithm 3. Watermark extraction.

If $\begin{cases} A' > B' \\ C' > D' \end{cases}$ **then extract "11" Else**

If $\begin{cases} A' > B' \\ C' \le D' \end{cases}$ **then extract "01" Else**

If $\begin{cases} A' \le B' \\ C' \le D' \end{cases}$ **then extract "00" Else**

extract "10"

VII. Performance Evaluation of the Proposed Method

In this article, four major criteria are applied to evaluate the newly proposed method.

1. Evaluation Parameters

The evaluation parameters are of three:

a) Payload: the maximum bits of watermark that can be embedded into one second of host signal.

b) Robustness against attacks: the BER parameter of this algorithm is obtained from dividing the number of errors by the total numbers of watermark bits.

c) Transparency: here the SNR and ODG are used to serve the purpose. Where the SNR is defined as follows

$$SNR(A,\tilde{A}) = 10\log_{10}\frac{\sum_{n=1}^{N} a^2(n)}{\sum_{n=1}^{N}(a(n) - \tilde{a}(n))^2} \qquad (6)$$

Where A represents an N-sample audio signal while $\tilde{A}$ is the watermarked version of that signal.

In this study, the ODG is computed by PEAQ[8] algorithm specified in ITU BS.1387-1. The calculated ODG values range from 0 to -4, where 0 means imperceptible, -1 stands for perceptible but not annoying, -2 represents mildly annoying, -3 means annoying and -4 stands for very annoying [7].

d) Time

2. Performance of the newly proposed method

As mentioned in section five, to embed watermark, at the beginning and the end of the frame, delimiters are attached to the embedded watermark. Next the Reed-Solomon ECC is applied on the watermark. If $|\text{delimiters}| = 16$, $l = |syn| = 16$ bits, n_{syn} =4 samples, the sampling rate= 44100, and the length of watermarks after Reed-Solomon(255,249) encoding becomes 104. The capacity can be obtained as follows:

[8] Perceptual Evaluation of Audio Quality

$$cap = \frac{56 - 16}{(16 \times 4 + 52 \times 512) \div 44100} \cong 66.09 \qquad (7)$$

Because of embedding the two bits of watermark into $l3{=}512$ samples of audio signal compared to [5] that embeds only one bit into 512 samples, the payload of proposed algorithm becomes double compared to the proposed algorithm in [5]. The SNR in this proposed method with different parameters α, β, m is compared with similar methods in Table 1. The results indicate that the SNR here, in comparison with [5] has not declined. Since the watermarking is stated in terms of SNR should be greater than 20 dB [4], this method has acceptable transparency for the various signals. The audio files are selected in such a manner that a set of different types of audio signals such as Rock, Pop, Classic, etc are included. The audio files which are used in evaluating the robustness of the method have the length of 10 second. The SNR and watermarking capacity in five different methods are compared in Table 2. Here the watermarking capacity is nearly doubled without any noticeable change in SNR.

The ODG with different parameters is compared with that of the method of [5] in Table 3. The column "#watermarks" represents the number of watermark bits in corresponding audio file. "Time" shows the amount of time for audio file. The results indicate that this method is imperceptible in different cases.

In this article, in order to assess the robustness of the proposed method, attacks such as MP3 128 kbps, additive noise, RC-low pass filtering and re-sampling are selected among the most damaging attacks. Although there is no standard for the implementation of the attacks, and here the attacks are implemented with the common tools, available in this field. The following attacks are performed on the watermarked signal:

MP3 128 kbps: this attack is implemented with the LAME encoder.

Additive noise: this attack is carried out using the StirMark benchmark.

RC-low pass filter: to implement this attack, the StirMark benchmark is used, where the filter cutoff frequency is 10 kHz.

Re-sampling: to implement this attack, the watermarked signal is sampled with sampling rate of 22050, and the obtained signal is sampled once more with a sampling rate of 44100. The BER is recorded in the result tables without ECC for both the methods. Tables 4, 5, 6, 7 and 8 show BER under MP3 compression, additive noise, low pass filtering, re-sampling and re-quantization attacks, respectively.

3. Timing Evaluation

As previously mentioned, time is the most important criterion in real-time watermarking. In this section, Time complexity of the proposed algorithm is compared with similar algorithms. The speed of synchronization detection which is the most time-consuming step in the watermarking process is calculated in different methods in term of samples per second(Table 9). The speed of these three methods is measured by means of MATLAB 2009 on a PC with the following specifications: 2.53 GHz processor, 3 GB RAM. The Watermark Embedding Time Complexity between [5] and the proposed method is compared in Table 10, above-mentioned experiment is performed in a situation the PC is only running MATLAB software.

Table 1. SNR for different methods and different parameters.

		SNR			
scheme		m=249	m=250	m=250	m=250
File	[5]	$\alpha = 4$ $\beta = 30$	$\alpha = 3$ $\beta = 7$	$\alpha = 3$ $\beta = 10$	$\alpha = 4$ $\beta = 10$
Floodplain	24.75	24.63	21.45	21.33	20.80
Stop Payment	22.88	22.53	21.21	21.09	20.72
Rust	26.55	25.15	22.42	22.32	21.81
The Firm	26.12	25.73	24.65	24.52	23.95
Sonati	21.84	23.39	23.13	23.11	23.05

Table 2. SNR and capacity of many schemes.

scheme	Content	sync	SNR(dB)	capacity
[8]	Short clip	yes	43.1	-
[9]	song	No	25	43
[5]	Song + quartet + SQAM	Yes	25.7	33.09
Proposed method	Song + quartet + SQAM	Yes	24.08	66.09

Table 3. ODG results

file	time	#watermarks	ODG [5]	ODG m=249 α=4 β=30	ODG m=250 α=3 β=10
Rust	2:33	117	-0.1	-0.025	-.131
Floodplain	3:13	146	0.0	-0.027	-0.03
Stop payment	2:09	99	0.0	-0.022	-0.116
The firm	0:10	8	0.0	0.0	0.0

Table 4. BER on MP3 128kbps

	BER%				
scheme		m=249	m=250	m=250	m=250
File	[5]	$\alpha = 4$ $\beta = 30$	$\alpha = 3$ $\beta = 7$	$\alpha = 3$ $\beta = 10$	$\alpha = 4$ $\beta = 10$
Floodplain	11.94	11.38	10.1	9.81	9.63
Stop Payment	11.94	15.62	12.77	12.89	11.44
Rust	14.03	18.87	17.13	15.04	16.6
The Firm	36.07	31.99	28.68	28.33	27.00
Sonati	23.54	30.37	27.52	28.54	29.38

Table 5. BER on additive noise.

	BER%				
scheme		m=249	m=250	m=250	m=250
File	[5]	$\alpha = 4$ $\beta = 30$	$\alpha = 3$ $\beta = 7$	$\alpha = 3$ $\beta = 10$	$\alpha = 4$ $\beta = 10$
Floodplain	1.39	3.94	1.97	2.26	1.8
Stop Payment	8.81	11.9	9.34	8.42	9.11
Rust	14.84	17.07	14.57	13.29	11.73
The Firm	20.99	21.19	19.39	19.22	19.68
Sonati	2.55	5.16	4.00	2.90	3.19

Table 6.BER on re-sampling

	BER%				
scheme		m=249	m=250	m=250	m=250
File	[5]	$\alpha = 4$ $\beta = 30$	$\alpha = 3$ $\beta = 7$	$\alpha = 3$ $\beta = 10$	$\alpha = 4$ $\beta = 10$
Floodplain	0	0	0	0	0
Stop Payment	0	0	0	0	0
Rust	2.43	5.11	2.72	2.72	2.55
The Firm	3.59	5.34	4.41	4.41	4.29
Sonati	0.11	0.05	0.05	0.05	0.05

Table 7. BER on RC-Low pass filter

scheme		BER%			
		m=249	m=250	m=250	m=250
File	[5]	$\alpha = 4$ $\beta = 30$	$\alpha = 3$ $\beta = 7$	$\alpha = 3$ $\beta = 10$	$\alpha = 4$ $\beta = 10$
Floodplain	1.27	4.23	2.20	1.91	1.97
Stop Payment	3.71	7.54	5.11	4.52	4.23
Rust	6.14	11.9	8.3	8.13	7.72
The Firm	6.84	10.97	10.22	9.93	10.10
Sonati	2.08	6.79	5.69	5.63	4.76

Table 8. BER on Re-quantization

scheme		BER%			
		m=249	m=250	m=250	m=250
File	[5]	$\alpha = 4$ $\beta = 30$	$\alpha = 3$ $\beta = 7$	$\alpha = 3$ $\beta = 10$	$\alpha = 4$ $\beta = 10$
Floodplain	0.003	0.007	0.006	0.005	0.005
Stop Payment	0.019	0.035	0.029	0.026	0.029
Rust	0.081	0.081	0.081	0.081	0.081
The Firm	0.13	0.16	0.14	0.14	0.14
Sonati	0.001	0.007	0.003	0.002	0.002

VIII. Hardware Simulation of Synchronization Code Detection Operation by FPGA

Time of synchronization code detection operation can be simulated by hardware. Simulation is made by FPGA as follows: it is assumed that the audio samples enter the FPGA from A/D converter or RAM. A vector with four members of 16-bit integers (v1), and four 16-bit vectors (v2, v3, v4, v5), are considered. There is a 16-bit vector (v6) where 16-bit barker code (synchronization code) is placed in it.

At each clock pulse, a 16-bit audio sample in v1 and v1 shifts one bit. After 4 clock pulses, this vector is filled and after that at every clock pulse the new sample will enter and old sample will be discarded. When v1 is filled, the synchronization detection process will begin. At first, the v1 shifts one bit in a manner that member 0 will be discarded. After that Sum of the intermediate samples (s1) and extreme samples (s2) is calculated, if s1 is bigger than s2, '1' will be substituted in the 15^{th} member in v2 otherwise '0' will be substituted. At this clock pulse, the v2 is compared with barker code; if they are equal the synchronization code is detected. At the next pulse a new sample will be placed in v1 and summation of intermediate and extreme samples would be calculated in the same manner; except that in this time extracted bit will be substituted in 15^{th} member of v3, and be compared with barker code .This procedure will continue and extracted bit in 3^{rd} and 4^{th} pulses would be concatenated to v4 and v5, respectively. In the 5^{th} clock pulse, after v1 is shifted, the extracted bit is concatenated to 0-14^{th} member of v1. This procedure will continue to the last audio signal sample, to find the multiple synchronization code, if any exists. The speed of synchronization detection process in the proposed algorithm on different types of FPGAs is presented in Table 11.

Table 9. Comparison of synchronization detection time complexity.

Watermarking algorithm	Synchronization detection speed (samples per second)	Consumed time per sample (s)
[6]	285.7	0.0035
[4]	107.13	0.0093
[5]	1010.1	0.00099
Proposed algorithm	4040.4	0.00025

Table 10. Comparison of watermark embedding time complexity

Watermarking algorithm	Watermark embedding (bps)	Time consumed per bit (s)
[5]	338.03	0.003
Proposed algorithm	1171.43	0.00085

Table 11. Speed of synchronization detection process in proposed algorithm on different types of FPGAs.

Family	Package	Device	Time per Sample(ns)
Spartan3	PQ208	XC3S50	11.65
Spartan6	TQG144	XC6SLX4	7.541
Virtex4	SF363	XC4VFX12	7.07
Virtex7	FFG1925	XC7V2000T	4.37

IX. Conclusion

A method for real-time audio watermarking with high payload is presented here. A method to achieve this objective is introduced in [5]. Although [5] is fast, the payload of the method is less than the similar works. Here, by introducing a new algorithm, the watermarking payload is doubled compared to [5]. The computational load is reduced by this method compared to [5]. Results of conducted evaluations where the most standard available evaluation tools are used indicate that this method has a proper robustness against signal processing attacks and in some cases shows a much better robustness compared to similar methods introduced previously. This method has a proper SNR and ODG; therefore it does not generate audible noise in host audio signal. This proposed method will be appropriate in practical and real-time applications.

References

References

[1] W. Lie and L. Chang, "Robust and high-quality time-domain audio watermarking based on low-frequency amplitude modification." *IEEE Trans. Multimedia*, vol. 8, no. 1, Feb. 2006, pp. 46-59.

[2] K. Yamamoto and M. Iwakiri, "Real-time audio watermarking with wavetable alternation in digital instrument." *Int. Conf. on Availability, Reliability and Security*, 2009, p. 786.

[3] J. Huang, Y. Wang and Y. Shi, "A blind audio watermarking algorithm with self-synchronization." *Proc. Circuits and Systems*, 2002, p. 627.

[4] S. Wu, J. Huang, D. Huang and Y. Shi, "Efficiently self-synchronized audio watermarking for assured audio data transmission." *IEEE Trans. Broadcasting*, vol. 51, no. 1, 2005, pp. 69-76.

[5] D. Megias, J. Serra-Ruiz and M. Fallahpour, "Efficient self-synchronized blind audio watermarking system based on time domain and FFT amplitude modification." *Signal Processing*, vol. 90, no. 12, Dec. 2010, pp. 3078- 3092.

[6] H. Nikmehr and S. T. Hashemy, "A new approach to audio watermarking using discrete wavelet and cosine transform." *Int. Conf. Communications Engineering*; 2010, p. 1.

[7] ITU-R. Recommendation BS.1387. *Method for objective measurements of perceived audio quality*, 1998.

[8] X. Y. Wang and H. Zhao, "A novel synchronization invariant audio watermarking scheme based on DWT and DCT." *IEEE Trans. Signal Processing*, vol. 56, no. 12, Dec. 2006, pp. 4835-4840.

[9] H. Kang, K. Yamaguchi, B. Kurkoski, K. Yamaguchi and K. Kobayashi, "Full-index-embedding patchwork algorithm for audio watermarking." *IEICE Trans. INF. & SYST*, vol. E91-D, no. 11, Nov. 2008, pp. 2731-2734.