United States Patent Application Publication	20250265293
Kind Code	A1
Publication Date	August 21, 2025
Inventor(s)	Srinivasan; Venugopal et al.

METHODS AND APPARATUS TO PERFORM AUDIO WATERMARKING AND WATERMARK DETECTION AND EXTRACTION

Abstract

Methods and apparatus to audio watermarking and watermark detection and extracted are described herein. An example method includes receiving a media content signal, sampling the media content signal to generate samples, storing the samples in a buffer, determining a first sequence of samples in the buffer, determining a second sequence of samples in the buffer, wherein the second sequence of samples is of substantially equal length as the first sequence of samples, calculating an average of the first sequence of samples and the second sequence of samples to generate an average sequence of samples, extracting an identifier from the average sequence of samples, and storing the identifier in a tangible memory.

Inventors: **Srinivasan; Venugopal (Palm Harbor, FL), Topchy; Alexander (New Port Richey, FL)**

Applicant: **The Nielsen Company (US), LLC** (New York, NY)

Family ID: **1000008578096**

Appl. No.: **18/986290**

Filed: **December 18, 2024**

Related U.S. Application Data

parent US continuation 18484792 20231011 parent-grant-document US 12189684 child US 18986290
parent US continuation 17676461 20220221 parent-grant-document US 11809489 child US 18484792
parent US continuation 16672128 20191101 parent-grant-document US 11256740 child US 17676461
parent US continuation 15698373 20170907 parent-grant-document US 10467286 child US

16672128
parent US continuation 13708578 20121207 ABANDONED child US 15698373
parent US continuation 12551220 20090831 parent-grant-document US 8359205 child US 13708578
us-provisional-application US 61174708 20090501
us-provisional-application US 61108380 20081024

Publication Classification

Int. Cl.: **G06F16/683** (20190101); **G11B20/10** (20060101); **H04H20/31** (20080101); **H04H60/37** (20080101); **H04H60/58** (20080101)

U.S. Cl.:

CPC **G06F16/683** (20190101); **G11B20/10** (20130101); **H04H20/31** (20130101); **H04H60/37** (20130101); **H04H60/58** (20130101); H04H2201/50 (20130101)

Background/Summary

CROSS-REFERENCE TO RELATED APPLICATIONS [0001] This disclosure is a continuation of U.S. patent application Ser. No. 18/484,792, filed Oct. 11, 2023, now issued as U.S. Pat. No. 12,189,684, which is a continuation of U.S. patent application Ser. No. 17/676,461, filed Feb. 21, 2022, now issued as U.S. Pat. No. 11,809,489, which is continuation of U.S. patent application Ser. No. 16/672,128, filed Nov. 1, 2019, now issued as U.S. Pat. No. 11,256,750, which is a continuation of U.S. patent application Ser. No. 15/698,373, filed Sep. 7, 2017, now issued as U.S. Pat. No. 10,467,286, which is a continuation of U.S. patent application Ser. No. 13/708,578, filed Dec. 7, 2012, which is a continuation of U.S. patent application Ser. No. 12/551,220, filed Aug. 31, 2009, now issued as U.S. Pat. No. 8,359,205, which claims the benefit of U.S. Provisional Application No. 61/174,708 filed May 1, 2009 and U.S. Provisional Application No. 61/108,380 filed Oct. 24, 2008, each of which is hereby incorporated by reference in its entirety.

TECHNICAL FIELD
[0002] The present disclosure relates generally to media monitoring and, more particularly, to methods and apparatus to perform audio watermarking and watermark detection and extraction.
BACKGROUND
[0003] Identifying media information and, more specifically, audio streams (e.g., audio information) is useful for assessing audience exposure to television, radio, or any other media. For example, in television audience metering applications, a code may be inserted into the audio or video of media, wherein the code is later detected at monitoring sites when the media is presented (e.g., played at monitored households). The information payload of the code/watermark embedded into original signal can consist of unique source identification, time of broadcast information, transactional information or additional content metadata.
[0004] Monitoring sites typically include locations such as, for example, households where the media consumption of audience members or audience member exposure to the media is monitored. For example, at a monitoring site, codes from the audio and/or video are captured and may be associated with audio or video streams of media associated with a selected channel, radio station, media source, etc. The collected codes may then be sent to a central data collection facility for

analysis. However, the collection of data pertinent to media exposure or consumption need not be limited to in-home exposure or consumption.

Description

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] FIG. **1** is a schematic depiction of a broadcast audience measurement system employing a program identifying code added to the audio portion of a composite television signal.

[0006] FIG. **2** is a block diagram of an example encoder of FIG. **1**.

[0007] FIG. **3** is a flow diagram illustrating an example encoding process that may be carried out by the example decoder of FIG. **2**.

[0008] FIG. **4** is a flow diagram illustrating an example process that may be carried to generate a frequency index table used in conjunction with the code frequency selector of FIG. **2**.

[0009] FIG. **5** is a chart illustrating critical band indices and how they correspond to short and long block sample indices.

[0010] FIG. **6** illustrates one example of selecting frequency components that will represent a particular information symbol.

[0011] FIGS. **7**-**9** are charts illustrating different example code frequency configurations that may be generated by the process of FIG. **4** and used in conjunction with the code frequency selector of FIG. **2**.

[0012] FIG. **10** illustrates the frequency relationship between the audio encoding indices.

[0013] FIG. **11** is a block diagram of the example decoder of FIG. **1**.

[0014] FIG. **12** is a flow diagram illustrating an example decoding process that may be carried out by the example encoder of FIG. **11**.

[0015] FIG. **13** is a flow diagram of an example process that may be carried out to stack audio in the decoder of FIG. **11**.

[0016] FIG. **14** is a flow diagram of an example process that may be carried out to determine a symbol encoded in an audio signal in the decoder of FIG. **11**.

[0017] FIG. **15** is a flow diagram of an example process that may be carried out to process a buffer to identify messages in the decoder of FIG. **11**.

[0018] FIG. **16** illustrates an example set of circular buffers that may store message symbols.

[0019] FIG. **17** illustrates an example set of pre-existing code flag circular buffers that may store message symbols.

[0020] FIG. **18** is a flow diagram of an example process that may be carried out to validate identified messages in the decoder of FIG. **11**.

[0021] FIG. **19** illustrates an example filter stack that may store identified messages in the decoder of FIG. **11**.

[0022] FIG. **20** is a schematic illustration of an example processor platform that may be used and/or programmed to perform any or all of the processes or implement any or all of the example systems, example apparatus and/or example methods described herein.

DETAILED DESCRIPTION

[0023] The following description makes reference to audio encoding and decoding that is also commonly known as audio watermarking and watermark detection, respectively. It should be noted that in this context, audio may be any type of signal having a frequency falling within the normal human audibility spectrum. For example, audio may be speech, music, an audio portion of an audio and/or video program or work (e.g., a television program, a movie, an Internet video, a radio program, a commercial spot, etc.), a media program, noise, or any other sound.

[0024] In general, as described in detail below, the encoding of the audio inserts one or more codes or information (e.g., watermarks) into the audio and, ideally, leaves the code inaudible to hearers of

the audio. However, there may be certain situations in which the code may be audible to certain listeners. The codes that are embedded in audio may be of any suitable length and any suitable technique for assigning the codes to information may be selected.

[0025] As described below, the codes or information to be inserted into the audio may be converted into symbols that will be represented by code frequency signals to be embedded in the audio to represent the information. The code frequency signals include one or more code frequencies, wherein different code frequencies or sets of code frequencies are assigned to represent different symbols of information. Techniques for generating one or more tables mapping symbols to representative code frequencies such that symbols are distinguishable from one another at the decoder are also described. Any suitable encoding or error correcting technique may be used to convert codes into symbols.

[0026] By controlling the amplitude at which these code frequency signals are input into the native audio, the presence of the code frequency signals can be imperceptible to human hearing. Accordingly, in one example, masking operations based on the energy content of the native audio at different frequencies and/or the tonality or noise-like nature of the native audio are used to provide information upon which the amplitude of the code frequency signals is based.

[0027] Additionally, it is possible that an audio signal has passed through a distribution chain, where, for example, the content has passed from a content originator to a network distributor (e.g., NBC national) and further passed to a local content distributor (e.g., NBC in Chicago). As the audio signal passes through the distribution chain, one of the distributors may encode a watermark into the audio signal in accordance with the techniques described herein, thereby including in the audio signal an indication of that distributors identity or the time of distribution. The encoding described herein is very robust and, therefore, codes inserted into the audio signal are not easily removed. Accordingly, any subsequent distributors of the audio content may use techniques described herein to encode the previously encoded audio signal in a manner such that the code of the subsequent distributor will be detectable and any crediting due that subsequent distributor will be acknowledged.

[0028] Additionally, due to the repetition or partial repetition of codes within a signal, code detection can be improved by stacking messages and transforming the encoded audio signal into a signal having an accentuated code. As the audio signal is sampled at a monitored location, substantially equal sized blocks of audio samples are summed and averaged. This stacking process takes advantage of the temporal properties of the audio signal to cause the code signal to be accentuated within the audio signal. Accordingly, the stacking process, when used, can provide increased robustness to noise or other interference. For example, the stacking process may be useful when the decoding operation uses a microphone that might pick up ambient noise in addition to the audio signal output by a speaker.

[0029] A further technique to add robustness to the decoding operations described herein provides for validation of messages identified by a decoding operation. After messages are identified in an encoded audio signal, the messages are added to a stack. Subsequent repetitions of messages are then compared to identify matches. When a message can be matched to another message identified at the proper repetition interval, the messages are identified as validated. When a message can be partially matched to another message that has already been validated, the message is marked as partially validated and subsequent messages are used to identify parts of the message that may have been corrupted. According to this example validation technique, messages are only output from the decoder if they can be validated. Such a technique prevents errors in messages caused by interference and/or detection errors.

[0030] The following examples pertain generally to encoding an audio signal with information, such as a code, and obtaining that information from the audio via a decoding process. The following example encoding and decoding processes may be used in several different technical applications to convey information from one place to another.

[0031] The example encoding and decoding processes described herein may be used to perform broadcast identification. In such an example, before a work is broadcast, that work is encoded to include a code indicative of the source of the work, the broadcast time of the work, the distribution channel of the work, or any other information deemed relevant to the operator of the system. When the work is presented (e.g., played through a television, a radio, a computing device, or any other suitable device), persons in the area of the presentation are exposed not only to the work, but, unbeknownst to them, are also exposed to the code embedded in the work. Thus, persons may be provided with decoders that operate on a microphone-based platform so that the work may be obtained by the decoder using free-field detection and processed to extract codes therefrom. The codes may then be logged and reported back to a central facility for further processing. The microphone-based decoders may be dedicated, stand-alone devices, or may be implemented using cellular telephones or any other types of devices having microphones and software to perform the decoding and code logging operations. Alternatively, wire-based systems may be used whenever the work and its attendant code may be picked up via a hard wired connection.
[0032] The example encoding and decoding processes described herein may be used, for example, in tracking and/or forensics related to audio and/or video works by, for example, marking copyrighted audio and/or associated video content with a particular code. The example encoding and decoding processes may be used to implement a transactional encoding system in which a unique code is inserted into a work when that work is purchased by a consumer. Thus, allowing a media distribution to identify a source of a work. The purchasing may include a purchaser physically receiving a tangible media (e.g., a compact disk, etc.) on which the work is included, or may include downloading of the work via a network, such as the Internet. In the context of transactional encoding systems, each purchaser of the same work receives the work, but the work received by each purchaser is encoded with a different code. That is, the code inserted in the work may be personal to the purchaser, wherein each work purchased by that purchaser includes that purchaser's code. Alternatively, each work may be may be encoded with a code that is serially assigned.
[0033] Furthermore, the example encoding and decoding techniques described herein may be used to carry out control functionality by hiding codes in a steganographic manner, wherein the hidden codes are used to control target devices programmed to respond to the codes. For example, control data may be hidden in a speech signal, or any other audio signal. A decoder in the area of the presented audio signal processes the received audio to obtain the hidden code. After obtaining the code, the target device takes some predetermined action based on the code. This may be useful, for example, in the case of changing advertisements within stores based on audio being presented in the store, etc. For example, scrolling billboard advertisements within a store may be synchronized to an audio commercial being presented in the store through the use of codes embedded in the audio commercial.
[0034] An example encoding and decoding system **100** is shown in FIG. **1**. The example system **100** may be, for example, a television audience measurement system, which will serve as a context for further description of the encoding and decoding processes described herein. The example system **100** includes an encoder **102** that adds a code or information **103** to an audio signal **104** to produce an encoded audio signal. The information **103** may be any selected information. For example, in a media monitoring context, the information **103** may be representative of an identity of a broadcast media program such as a television broadcast, a radio broadcast, or the like. Additionally, the information **103** may include timing information indicative of a time at which the information **103** was inserted into audio or a media broadcast time. Alternatively, the code may include control information that is used to control the behavior of one or more target devices.
[0035] The audio signal **104** may be any form of audio including, for example, voice, music, noise, commercial advertisement audio, audio associated with a television program, live performance, etc. In the example of FIG. **1**, the encoder **102** passes the encoded audio signal to a transmitter **106**. The

transmitter **106** transmits the encoded audio signal along with any video signal **108** associated with the encoded audio signal. While, in some instances, the encoded audio signal may have an associated video signal **108**, the encoded audio signal need not have any associated video.
[0036] In one example, the audio signal **104** is a digitized version of an analog audio signal, wherein the analog audio signal has been sampled at 48 kilohertz (KHz). As described below in detail, two seconds of audio, which correspond to 96,000 audio samples at the 48 KHz sampling rate, may be used to carry one message, which may be a synchronization message and 49 bits of information. Using an encoding scheme of 7 bits per symbol, the message requires transmission of eight symbols of information. Alternatively, in the context of overwriting described below, one synchronization symbol is used and one information symbol conveying one of 128 states follows the synchronization symbol. As described below in detail, according to one example, one 7-bit symbol of information is embedded in a long block of audio samples, which corresponds to 9216 samples. In one example, such a long block includes 36 overlapping short blocks of 256 samples, wherein in a 50% overlapping block **256** of the samples are old and 256 samples are new.
[0037] Although the transmit side of the example system **100** shown in FIG. **1** shows a single transmitter **106**, the transmit side may be much more complex and may include multiple levels in a distribution chain through which the audio signal **104** may be passed. For example, the audio signal **104** may be generated at a national network level and passed to a local network level for local distribution. Accordingly, although the encoder **102** is shown in the transmit lineup prior to the transmitter **106**, one or more encoders may be placed throughout the distribution chain of the audio signal **104**. Thus, the audio signal **104** may be encoded at multiple levels and may include embedded codes associated with those multiple levels. Further details regarding encoding and example encoders are provided below.
[0038] The transmitter **106** may include one or more of a radio frequency (RF) transmitter that may distribute the encoded audio signal through free space propagation (e.g., via terrestrial or satellite communication links) or a transmitter used to distribute the encoded audio signal through cable, fiber, etc. In one example, the transmitter **106** may be used to broadcast the encoded audio signal throughout a broad geographical area. In other cases, the transmitter **106** may distribute the encoded audio signal through a limited geographical area. The transmission may include up-conversion of the encoded audio signal to radio frequencies to enable propagation of the same. Alternatively, the transmission may include distributing the encoded audio signal in the form of digital bits or packets of digital bits that may be transmitted over one or more networks, such as the Internet, wide area networks, or local area networks. Thus, the encoded audio signal may be carried by a carrier signal, by information packets or by any suitable technique to distribute the audio signals.
[0039] When the encoded audio signal is received by a receiver **110**, which, in the media monitoring context, may be located at a statistically selected metering site **112**, the audio signal portion of the received program signal is processed to recover the code, even though the presence of that code is imperceptible (or substantially imperceptible) to a listener when the encoded audio signal is presented by speakers **114** of the receiver **110**. To this end, a decoder **116** is connected either directly to an audio output **118** available at the receiver **110** or to a microphone **120** placed in the vicinity of the speakers **114** through which the audio is reproduced. The received audio signal can be either in a monaural or stereo format. Further details regarding decoding and example decoders are provided below.
Audio Encoding
[0040] As explained above, the encoder **102** inserts one or more inaudible (or substantially inaudible) codes into the audio **104** to create encoded audio. One example encoder **102** is shown in FIG. **2**. In one implementation, the example encoder **102** of FIG. **2** may be implemented using, for example, a digital signal processor programmed with instructions to implement an encoding lineup **202**, the operation of which is affected by the operations of a prior code detector **204** and a masking

lineup **206**, either or both of which can be implemented using a digital signal processor programmed with instructions. Of course, any other implementation of the example encoder **102** is possible. For example, the encoder **102** may be implemented using one or more processors, programmable logic devices, or any suitable combination of hardware, software, and firmware.
[0041] In general, during operation, the encoder **102** receives the audio **104** and the prior code detector **204** determines if the audio **104** has been previously encoded with information, which will make it difficult for the encoder **102** to encode additional information into the previously encoded audio. For example, a prior encoding may have been performed at a prior location in the audio distribution chain (e.g., at a national network level). The prior code detector **204** informs the encoding lineup **202** as to whether the audio has been previously encoded. The prior code detector **204** may be implemented by a decoder as described herein.
[0042] The encoding lineup **202** receives the information **103** and produces code frequency signals based thereon and combines the code frequency signal with the audio **104**. The operation of the encoding lineup **202** is influenced by the output of the prior code detector **204**. For example, if the audio **104** has been previously encoded and the prior code detector **204** informs the encoding lineup **202** of this fact, the encoding lineup **202** may select an alternate message that is to be encoded in the audio **104** and may also alter the details by which the alternate message is encoded (e.g., different temporal location within the message, different frequencies used to represent symbols, etc.).
[0043] The encoding lineup **202** is also influenced by the masking lineup **206**. In general, the masking lineup **206** processes the audio **104** corresponding to the point in time at which the encoding lineup **202** wants to encode information and determines the amplitude at which the encoding should be performed. As described below, the masking lineup **206** may output a signal to control code frequency signal amplitudes to keep the code frequency signal below the threshold of human perception.
[0044] As shown in the example of FIG. **2**, the encoding lineup includes a message generator **210**, a symbol selector **212**, a code frequency selector **214**, a synthesizer **216**, an inverse Fourier transform **218**, and a combiner **220**. The message generator **210** is responsive to the information **103** and outputs messages having the format generally shown at reference numeral **222**. The information **103** provided to the message generator may be the current time, a television or radio station identification, a program identification, etc. In one example, the message generator **210** may output a message every two seconds. Of course, other messaging intervals are possible.
[0045] In one example, the message format **222** representative of messages output from the message generator **210** includes a synchronization symbol **224**. The synchronization symbol **224** is used by decoders, examples of which are described below, to obtain timing information indicative of the start of a message. Thus, when a decoder receives the synchronization symbol **224**, that decoder expects to see additional information following the synchronization symbol **224**.
[0046] In the example message format **222** of FIG. **2**, the synchronization symbol **224**, is followed by 42 bits of message information **226**. This information may include a binary representation of a station identifier and coarse timing information. In one example, the timing information represented in the 42 bits of message information **226** changes every 64 seconds, or 32 message intervals. Thus, the 42 bits of message information **226** remain static for 64 seconds. The seven bits of message information **228** may be high resolution time that increments every two seconds.
[0047] The message format **222** also includes pre-existing code flag information **230**. However, the pre-existing code flag information **230** is only selectively used to convey information. When the prior code detector **204** informs the message generator **210** that the audio **104** has not been previously encoded, the pre-existing code flag information **230** is not used. Accordingly, the message output by the message generator only includes the synchronization symbol **224**, the 42 bits of message information **226**, and the seven bits of message information **228**; the pre-existing code flag information **230** is blank or filled by unused symbol indications. In contrast, when the prior

code detector **204** provides to the message generator **210** an indication that the audio **104** into which the message information is to be encoded has previously been encoded, the message generator **210** will not output the synchronization symbol **224**, the 42 bits of message information **226**, or the seven bits of message information **228**. Rather, the message generator **210** will utilize only the pre-existing code flag information **230**. In one example, the pre-existing code flag information will include a pre-existing code flag synchronization symbol to signal that pre-existing code flag information is present. The pre-existing code flag synchronization symbol is different from the synchronization symbol **224** and, therefore, can be used to signal the start of pre-existing code flag information. Upon receipt of the pre-existing code flag synchronization symbol, a decoder can ignore any prior-received information that aligned in time with a synchronization symbol **224**, 42 bits of message information **226**, or seven bits of message information **228**. To convey information, such as a channel indication, a distribution identification, or any other suitable information, a single pre-existing code flag information symbol follows the pre-existing code flag synchronization symbol. This pre-existing code flag information may be used to provide for proper crediting in an audience monitoring system.

[0048] The output from the message generator **210** is passed to the symbol selector **212**, which selects representative symbols. When the synchronization symbol **224** is output, the symbol selector may not need to perform any mapping because the synchronization symbol **224** is already in symbol format. Alternatively, if bits of information are output from the message generator **210**, the symbol selector may use straight mapping, wherein, for example seven bits output from the message generator **210** are mapped to a symbol having the decimal value of the seven bits. For example, if a value of 1010101 is output from the message generator **210**, the symbol selector may map those bits to the symbol **85**. Of course other conversions between bits and symbols may be used. In certain examples, redundancy or error encoding may be used in the selection of symbols to represent bits. Additionally, any other suitable number of bits than seven may be selected to be converted into symbols. The number of bits used to select the symbol may be determined based on the maximum symbol space available in the communication system. For example, if the communication system can only transmit one of four symbols at a time, then only two bits from the message generator **210** would be converted into symbols at a time.

[0049] The symbols from the symbol selector **212** are passed to the code frequency selector **214** that selects code frequencies that are used to represent the symbol. The symbol selector **212** may include one or more look up tables (LUTs) **232** that may be used to map the symbols into code frequencies that represent the symbols. That is, a symbol is represented by a plurality of code frequencies that the encoder **102** emphasizes in the audio to form encoded audio that is transmitted. Upon receipt of the encoded audio, a decoder detects the presence of the emphasized code frequencies and decodes the pattern of emphasized code frequencies into the transmitted symbol. Thus, the same LUT selected at the encoder **210** for selecting the code frequencies needs to be used in the decoder. One example LUT is described in conjunction with FIGS. **3**-**5**. Additionally, example techniques for generating LUTs are provided in conjunction with FIGS. **7**-**9**.

[0050] The code frequency selector **214** may select any number of different LUTs depending of various criteria. For example, a particular LUT or set of LUTs may be used by the code frequency selector **214** in response to the prior receipt of a particular synchronization symbol. Additionally, if the prior code detector **204** indicates that a message was previously encoded into the audio **104**, the code frequency selector **214** may select a lookup table that is unique to pre-existing code situations to avoid confusion between frequencies used to previously encode the audio **104** and the frequencies used to include the pre-existing code flag information.

[0051] An indication of the code frequencies that are selected to represent a particular symbol is provided to the synthesizer **216**. The synthesizer **216** may store, for each short block constituting a long block, three complex Fourier coefficients representative of each of the possible code frequencies that the code frequency selector **214** will indicate. These coefficients represent the

transform of a windowed sinusoidal code frequency signal whose phase angle corresponds to the starting phase angle of code sinusoid in that short block.

[0052] While the foregoing describes an example code synthesizer **216** that generates sine waves or data representing sine waves, other example implementations of code synthesizers are possible. For example, rather than generating sine waves, another example code synthesizer **216** may output Fourier coefficients in the frequency domain that are used to adjust amplitudes of certain frequencies of audio provided to the combiner combiner **220**. In this manner, the spectrum of the audio may be adjusted to include the requisite sine waves.

[0053] The three complex amplitude-adjusted Fourier coefficients corresponding to the symbol to be transmitted are provided from the synthesizer **216** to the inverse Fourier transform **218**, which converts the coefficients into time-domain signals having the prescribed frequencies and amplitudes to allow their insertion into the audio to convey the desired symbols are coupled to the combiner **220**. The combiner **220** also receives the audio. In particular, the combiner **220** inserts the signals from the inverse Fourier transform **218** into one long block of audio samples. As described above, for a given sampling rate of 48 KHz, a long block is 9216 audio samples. In the provided example, the synchronization symbol and 49 bits of information require a total of eight long blocks. Because each long block is 9216 audio samples, only 73,728 samples of audio **104** are needed to encode a given message. However, because messages begin every two seconds, which is every 96,000 audio samples, there will be many samples at the end of the 96,000 audio samples that are not encoded. The combining can be done in the digital domain, or in the analog domain.

[0054] However, in the case of a pre-existing code flag, the pre-existing code flag is inserted into the audio **104** after the last symbol representing the previously inserted seven bits of message information. Accordingly, insertion of the pre-existing code flag information begins at sample 73,729 and runs for two long blocks, or 18,432 samples. Accordingly, when pre-existing code flag information is used, fewer of the 96,000 audio samples **104** will be unencoded.

[0055] The masking lineup **206** includes an overlapping short block maker that makes short blocks of 512 audio samples, wherein 256 of the samples are old and 256 samples are new. That is, the overlapping short block maker **240** makes blocks of 512 samples, wherein 256 samples are shifted into or out of the buffer at one time. For example, when a first set of 256 samples enters the buffer, the oldest 256 samples are shifted out of the buffer. On a subsequent iteration, the first set of 256 samples are shifted to a latter position of the buffer and 256 samples are shifted into the buffer. Each time a new short block is made by shifting in 256 new samples and removing the 256 oldest samples, the new short block is provided to a masking evaluator **242**. The **512** sample block output from the overlapping short block maker **240** is multiplied by a suitable window function such that an “overlap-and-add” operation will restore the audio samples to their correct value at the output. A synthesized code signal to be added to an audio signal is also similarly windowed to prevent abrupt transitions at block edges when there is a change in code amplitude from one 512-sample block to the next overlapped 512-sample block. These transitions if present create audible artifacts.

[0056] The masking evaluator **242** receives samples of the overlapping short block (e.g., 512 samples) and determines an ability of the same to hide code frequencies to human hearing. That is, the masking evaluator determines if code frequencies can be hidden within the audio represented by the short block by evaluating each critical band of the audio as a whole to determine its energy and determining the noise-like or tonal-like attributes of each critical band and determining the sum total ability of the critical bands to mask the code frequencies. According to the illustrated example, the bandwidth of the critical bands increases with frequency. If the masking evaluator **242** determines that code frequencies can be hidden in the audio **104**, the masking evaluator **204** indicates the amplitude levels at which the code frequencies can be inserted within the audio **104**, while still remaining hidden and provides the amplitude information to the synthesizer **216**.

[0057] In one example, the masking evaluator **242** conducts the masking evaluation by determining a maximum change in energy Eb or a masking energy level that can occur at any critical frequency

band without making the change perceptible to a listener. The masking evaluation carried out by the masking evaluator **242** may be carried out as outlined in the Moving Pictures Experts Group-Advanced Audio Encoding (MPEG-AAC) audio compression standard ISO/IEC 13818-7:1997, for example. The acoustic energy in each critical band influences the masking energy of its neighbors and algorithms for computing the masking effect are described in the standards document such as ISO/IEC 13818-7:1997. These analyses may be used to determine for each short block the masking contribution due to tonality (e.g., how much the audio being evaluated is like a tone) as well as noise like (i.e., how much the audio being evaluated is like noise) features. Further analysis can evaluate temporal masking that extends masking ability of the audio over short time, typically, for 50-100 milliseconds (ms). The resulting analysis by the masking evaluator **242** provides a determination, on a per critical band basis, the amplitude of a code frequency that can be added to the audio **104** without producing any noticeable audio degradation (e.g., without being audible).

[0058] Because a **256** sample block will appear in both the beginning of one short block and the end of the next short block and, thus, will be evaluated two times by the masking evaluator **242**, the masking evaluator makes two masking evaluations including the **256** sample block. The amplitude indication provided to the synthesizer **216** is a composite of those two evaluations including that 256 sample block and the amplitude indication is timed such that the amplitude of the code inserted into the 256 samples is timed with those samples arriving at the combiner **220**.

[0059] Referring now to FIGS. **3-5**, an example LUT **232** is shown that includes one column representing symbols **302** and seven columns **304**, **306**, **308**, **310**, **312**, **314**, **316** representing numbered code frequency indices. The LUT **232** includes 128 rows, which are used to represent data symbols. Because the LUT **232** includes 128 different data symbols, data may be sent at a rate of seven bits per symbol. The frequency indices in the table may range from 180-656 and are based on a long block size of 9216 samples and a sampling rate of 48 KHz. Accordingly, the frequencies corresponding to these indices range between 937.5 Hz and 3126.6 Hz, which falls into the humanly audible range. Of course, other sampling rates and frequency indices may be selected. A description of a process to generate a LUT, such as the table **232** is provided in conjunction with FIGS. **7-9**.

[0060] In one example operation of the code frequency selector **214**, a symbol of **25** (e.g., a binary value of 0011001) is received from the symbol selector **212**. The code frequency selector **214** accesses the LUT **232** and reads row **25** of the symbol column **302**. From this row, the code frequency selector reads that code frequency indices **217**, **288**, **325**, **403**, **512**, **548**, and **655** are to be emphasized in the audio **104** to communicate the symbol **25** to the decoder. The code frequency selector **214** then provides an indication of these indices to the synthesizer **216**, which synthesizes the code signals by outputting Fourier coefficients corresponding to these indices.

[0061] The combiner **220** receives both the output of the code synthesizer **216** and the audio **104** and combines them to form encoded audio. The combiner **220** may combine the output of the code synthesizer **216** and the audio **104** in an analog or digital form. If the combiner **220** performs a digital combination, the output of the code synthesizer **216** may be combined with the output of a sampler, rather than the audio that is input to the sampler. For example, the audio block in digital form may be combined with the sine waves in digital form. Alternatively, the combination may be carried out in the frequency domain, wherein frequency coefficients of the audio are adjusted in accordance with frequency coefficients representing the sine waves. As a further alternative, the sine waves and the audio may be combined in analog form. The encoded audio may be output from the combiner **220** in analog or digital form. If the output of the combiner **220** is digital, it may be subsequently converted to analog form before being coupled to the transmitter **106**.

[0062] An example encoding process **600** is shown in FIG. **6**. The example process **600** may be carried out by the example encoder **102** of FIG. **2**, or by any other suitable encoder. The example process **600** begins when audio samples to be encoded are received (block **602**). The process **600** then determines if the received samples have been previously encoded (block **604**). This

determination may be carried out, for example, by the prior code detector **204** of FIG. **2**, or by any suitable decoder configured to examine the audio to be encoded for evidence of a prior encoding.
[0063] If the received samples have not been previously encoded (block **604**), the process **600** generates a communication message (block **606**), such as a communication message having the format shown in FIG. **2** at reference numeral **222**. In one particular example, when the audio has not been previously encoded, the communication message may include a synchronization portion and one or more portions including data bits. The communication message generation may be carried out, for example, by the message generator **210** of FIG. **2**.
[0064] The communication message is then mapped into symbols (block **608**). For example, the synchronization information need not be mapped into a symbol if the synchronization information is already a symbol. In another example, if a portion of the communication message is a series of bits, such bits or groups of bits may be represented by one symbol. As described above in conjunction with the symbol selector **212**, which is one manner in which the mapping (block **608**) may be carried out, one or more tables or encoding schemes may be used to convert bits into symbols. For example, some techniques may include the use of error correction coding, or the like, to increase message robustness through the use of coding gain. In one particular example implementation having a symbol space sized to accommodate 128 data symbols, seven bits may be converted into one symbol. Of course, other numbers of bits may be processed depending on many factors including available symbol space, error correction encoding, etc.
[0065] After the communication symbols have been selected (block **608**), the process **600** selects a LUT that will be used to determine the code frequencies that will be used to represent each symbol (block **610**). In one example, the selected LUT may be the example LUT **232** of FIGS. **3**-**5**, or may be any other suitable LUT. Additionally, the LUT may be any LUT generated as described in conjunction with FIGS. **7**-**9**. The selection of the LUT may be based on a number of factors including the synchronization symbol that is selected during the generation of the communication message (block **606**).
[0066] After the symbols have been generated (block **608**) and the LUT is selected (block **610**), the symbols are mapped into code frequencies using the selected LUT (block **612**). In one example in which the LUT **232** of FIG. **3**-**5** is selected, a symbol of, for example, 35 would be mapped to the frequency indices **218**, **245**, **360**, **438**, **476**, **541**, and **651**. The data space in the LUT is between symbol **0** and symbol **127** and symbol **128**, which uses a unique set of code frequencies that do not match any other code frequencies in the table, is used to indicate a synchronization symbol. The LUT selection (block **610**) and the mapping (block **612**) may be carried out by, for example, the code frequency selector **214** of FIG. **2**. After the code frequencies are selected, an indication of the same is provided to, for example, the synthesizer **216** of FIG. **2**.
[0067] Code signals including the code frequencies are then synthesized (block **614**) at amplitudes according to a masking evaluation, which is described in conjunction with blocks **240** and **242** or FIG. **2**, and is described in conjunction with the process **600** below. In one example, the synthesis of the code frequency signals may be carried out by providing appropriately scaled Fourier coefficients to an inverse Fourier process. In one particular example, three Fourier coefficients may be output to represent each code frequency in the code frequency signals. Accordingly, the code frequencies may be synthesized by the inverse Fourier process in a manner in which the synthesized frequencies are windowed to prevent spill over into other portions of the signal into which the code frequency signals are being embedded. One example configuration that may be used to carry out the synthesis of block **614** is shown at blocks **216** and **218** of FIG. **2**. Of course other implementations and configurations are possible.
[0068] After the code signals including the code frequencies have been synthesized, they are combined with the audio samples (block **616**). As described in conjunction with FIG. **2**, the combination of the code signals and the audio is such that one symbol is inserted into each long block of audio samples. Accordingly, to communicate one synchronization symbol and 49 data bits,

information is encoded into eight long blocks of audio information: one long block for the synchronization symbol and one long block for each seven bits of data (assuming seven bits/symbol encoding). The messages are inserted into the audio at two second intervals. Thus, the eight long blocks of audio immediately following the start of a message may be encoded with audio and the remaining long blocks that make up the balance of the two second of audio may be unencoded.

[0069] The insertion of the code signal into the audio may be carried out by adding samples of the code signal to samples of the host audio signal, wherein such addition is done in the analog domain or in the digital domain. Alternatively, with proper frequency alignment and registration, frequency components of the audio signal may be adjusted in the frequency domain and the adjusted spectrum converted back into the time domain.

[0070] The foregoing described the operation of the process **600** when the process determined that the received audio samples have not been previously encoded (block **604**). However, in situations in which a portion of media has been through a distribution chain and encoded as it was processed, the received samples of audio processed at block **604** already include codes. For example, a local television station using a courtesy news clip from CNN in a local news broadcast might not get viewing credit based on the prior encoding of the CNN clip. As such, additional information is added to the local news broadcast in the form of pre-existing code flag information. If the received samples of audio have been previously encoded (block **604**), the process generates pre-existing code flag information (block **618**). The pre-existing code flag information may include the generation of an pre-existing code flag synchronization symbol and, for example, the generation of seven bits of data, which will be represented by a single data symbol. The data symbol may represent a station identification, a time, or any other suitable information. For example, a media monitoring site (MMS) may be programmed to detect the pre-existing code flag information to credit the station identified therein.

[0071] After the pre-existing code flag information has been generated (block **618**), the process **600** selects the pre-existing code flag LUT that will be used to identify code frequencies representative of the pre-existing code flag information (block **620**). In one example, the pre-existing code flag LUT may be different than other LUTs used in non-pre-existing code conditions. In one particular example, the pre-existing code flag synchronization symbol may be represented by the code frequencies **220**, **292**, **364**, **436**, **508**, **580**, and **652**.

[0072] After the pre-existing code flag information is generated (block **618**) and the pre-existing code flag LUT is selected (block **620**), the pre-existing code flag symbols are mapped to code frequencies (block **612**), and the remainder of the processing follows as previously described.

[0073] Sometime before the code signal is synthesized (block **614**), the process **600** conducts a masking evaluation to determine the amplitude at which the code signal should be generated so that it still remains inaudible or substantially inaudible to human hearers. Accordingly, the process **600** generates overlapping short blocks of audio samples, each containing 512 audio samples (block **622**). As described above, the overlapping short blocks include 50% old samples and 50% newly received samples. This operation may be carried out by, for example, the overlapping short block maker **240** of FIG. **2**.

[0074] After the overlapping short blocks are generated (block **622**), masking evaluations are performed on the short blocks (block **624**). For example, this may be carried out as described in conjunction with block **242** of FIG. **2**. The results of the masking evaluation are used by the process **600** at block **614** to determine the amplitude of the code signal to be synthesized. The overlapping short block methodology may yield two masking evaluation for a particular 256 samples of audio (one when the 256 samples are the “new samples,” and one when the 256 samples are the “old samples”), the result provided to block **614** of the process **600** may be a composite of these masking evaluations. Of course, the timing of the process **600** is such that the masking evaluations for a particular block of audio are used to determine code amplitudes for that block of

audio.

Lookup Table Generation

[0075] A system **700** for populating one or more LUTs with code frequencies corresponding to symbols may be implemented using hardware, software, combinations of hardware and software, firmware, or the like. The system **700** of FIG. **7** may be used to generate any number of LUTs, such as the LUT of FIGS. **3-5**. The system **700** which operates as described below in conjunction with FIG. **7** and FIG. **8**, results in a code frequency index LUT, wherein: (1) two symbols of the table are represented by no more than one common frequency index, (2) not more than one of the frequency indices representing a symbol reside in one audio critical band as defined by the MPEG-AA compression standard ISO/IEC 13818-7:1997, and (3) code frequencies of neighboring critical bands are not used to represent a single symbol. Criteria number **3** helps to ensure that audio quality is not compromised during the audio encoding process.

[0076] A critical band pair definer **702** defines a number (P) of critical band pairs. For example, referring to FIG. **9**, a table **900** includes columns representing AAC critical band indices **902**, short block indices **904** in the range of the AAC indices, and long block indices **906** in the range of the AAC indices. In one example, the value of P may be seven and, thus, seven critical band pairs are formed from the AAC indices (block **802**). FIG. **10** shows the frequency relationship between the AAC indices. According to one example, as shown at reference numeral **1002** in FIG. **10** wherein frequencies of critical band pairs are shown as separated by dotted lines, AAC indices may be selected into pairs as follows: five and six, seven and eight, nine and ten, eleven and twelve, thirteen and fourteen, fifteen and sixteen, and seventeen and seventeen. The AAC index of seventeen includes a wide range of frequencies and, therefore, index **17** is shown twice, once for the low portion and once for the high portion.

[0077] A frequency definer **704** defines a number of frequencies (N) that are selected for use in each critical band pair. In one example, the value of N is sixteen, meaning that there are sixteen data positions in the combination of the critical bands that form each critical band pair. Reference numeral **1004** in FIG. **10** identifies the seventeen frequency positions are shown. The circled position four is reserved for synchronization information and, therefore, is not used for data.

[0078] A number generator **706** defines a number of frequency positions in the critical band pairs defined by the critical band pair definer **702**. In one example the number generator **706** generates all NP, P-digit numbers. For example, if N is 16 and P is 7, the process generates the numbers 0 through 268435456, but may do so in base 16-hexadecimal, which would result in the values 0 through 10000000.

[0079] A redundancy reducer **708** then eliminates all number from the generated list of numbers sharing more than one common digit between them in the same position. This ensures compliance with criteria (1) above because, as described below, the digits will be representative of the frequencies selected to represent symbols. An excess reducer **710** may then further reduce the remaining numbers from the generated list of numbers to the number of needed symbols. For example, if the symbol space is 129 symbols, the remaining numbers are reduced to a count of 129. The reduction may be carried out at random, or by selecting remaining numbers with the greatest Euclidean distance, or my any other suitable data reduction technique. In another example, the reduction may be carried out in a pseudorandom manner.

[0080] After the foregoing reductions, the count of the list of numbers is equal to the number of symbols in the symbol space. Accordingly, a code frequency definer **712** defines the remaining numbers in base P format to represent frequency indices representative of symbols in the critical band pairs. For example, referring to FIG. **10**, the hexadecimal number F1E4B0F is in base **16**, which matches P. The first digit of the hexadecimal number maps to a frequency component in the first critical band pair, the second digit to the second critical band pair, and so on. Each digit represents the frequency index that will be used to represent the symbol corresponding to the hexadecimal number F1E4B0F.

[0081] Using the first hexadecimal number as an example of mapping to a particular frequency index, the decimal value of Fh is 15. Because position four of each critical band pair is reserved for non-data information, the value of any hexadecimal digit greater than four is incremented by the value of one decimal. Thus, the **15** becomes a **16**. The **16** is thus designated (as shown with the asterisk in FIG. **10**) as being the code frequency component in the first critical band pair to represent the symbol corresponding to the hexadecimal number F1E4B0F. Though not shown in FIG. **10**, the index **1** position (e.g., the second position from the far left in the critical band **7** would be used to represent the hexadecimal number F1E4B0F.
[0082] A LUT filler **714** receives the symbol indications and corresponding code frequency component indications from the code frequency definer **712** and fills this information into a LUT.
[0083] An example code frequency index table generation process **800** is shown in FIG. **8**. The process **800** may be implemented using the system of FIG. **7**, or any other suitable configuration. The process **800** of FIG. **8** may be used to generate any number of LUTs, such as the LUT of FIGS. **3-5**. While one example process **800** is shown, other processes may be used. The result of the process **800** is a code frequency index LUT, wherein: (1) two symbols of the table are represented by no more than one common frequency index, (2) not more than one of the frequency indices representing a symbol reside in one audio critical band as defined by the MPEG-AA compression standard ISO/IEC 13818-7:1997, and (3) code frequencies of neighboring critical bands are not used to represent a single symbol. Criteria number 3 helps to ensure that audio quality is not compromised during the audio encoding process.
[0084] The process **800** begins by defining a number (P) of critical band pairs. For example, referring to FIG. **9**, a table **900** includes columns representing AAC critical band indices **902**, short block indices **904** in the range of the AAC indices, and long block indices **906** in the range of the AAC indices. In one example, the value of P may be seven and, thus, seven critical band pairs are formed from the AAC indices (block **802**). FIG. **10** shows the frequency relationship between the AAC indices. According to one example, as shown at reference numeral **1002** in FIG. **10** wherein frequencies of critical band pairs are shown as separated by dotted lines, AAC indices may be selected into pairs as follows: five and six, seven and eight, nine and ten, eleven and twelve, thirteen and fourteen, fifteen and sixteen, and seventeen and seventeen. The AAC index of seventeen includes a wide range of frequencies and, therefore, index **17** is shown twice, once for the low portion and once for the high portion.
[0085] After the band pairs have been defined (block **802**), a number of frequencies (N) is selected for use in each critical band pair (block **804**). In one example, the value of N is sixteen, meaning that there are sixteen data positions in the combination of the critical bands that form each critical band pair. As shown in FIG. **10** as reference numeral **1004**, the seventeen frequency positions are shown. The circled position four is reserved for synchronization information and, therefore, is not used for data.
[0086] After the number of critical band pairs and the number of frequency positions in the pairs is defined, the process **800** generates all NP, P-digit numbers with no more than one hexadecimal digit in common (block **806**). For example, if N is 16 and P is 7, the process generates the numbers 0 through 268435456, but may do so in base 16-hexadecimal, which would results in 0 through FFFFFFF, but does not include the numbers that share more than one common hexadecimal digit. This ensures compliance with criteria (1) above because, as described below, the digits will be representative of the frequencies selected to represent symbols.
[0087] According to an example process for determining a set of numbers that comply with criteria (1) above (and any other desired criteria), the numbers in the range from 0 to NP-1 are tested. First, the value corresponding to zero is stored as the first member of the result set R. Then, the numbers from 1 to NP-1 are selected for analysis to determine if they meet criteria (1) when compared to the members of R. Each number that meets criteria (1) when compared against all the current entries in R is added to the result set. In particular, according to the example process, in order to test a

number K, each hexadecimal digit of interest in K is compared to the corresponding hexadecimal digit of interest in an entry M from the current result set. In the 7 comparisons not more than one hexadecimal digit of K should equal the corresponding hexadecimal digit of M. If, after comparing K against all numbers currently in the result set, no member of the latter has more than one common hexadecimal digit, then K is added to the result set R. The algorithm iterates through the set of possible numbers until all values meeting criteria (1) have been identified.

[0088] While the foregoing describes an example process for determining a set of numbers that meets criteria (1), any process or algorithm may be used and this disclosure is not limited to the process described above. For example, a process may use heuristics, rules, etc. to eliminate numbers from the set of numbers before iterating throughout the set. For example, all of the numbers where the relevant bits start with two 0's, two 1's, two 2's, etc. and end with two 0's, two 1's, two 2's, etc. could immediately be removed because they will definitely have a hamming distance less than 6. Additionally or alternatively, an example process may not iterate through the entire set of possible numbers. For example, a process could iterate until enough numbers are found (e.g., 128 numbers when 128 symbols are desired). In another implementation, the process may randomly select a first value for inclusion in the set of possible values and then may search iteratively or randomly through the remaining set of numbers until a value that meets the desired criteria (e.g., criteria (1)) is found.

[0089] The process **800** then selects the desired numbers from the generated values (block **810**). For example, if the symbol space is 129 symbols, the remaining numbers are reduced to a count of **129**. The reduction may be carried out at random, or by selecting remaining numbers with the greatest Euclidean distance, or my any other suitable data reduction technique.

[0090] After the foregoing reductions, the count of the list of numbers is equal to the number of symbols in the symbol space. Accordingly, the remaining numbers in base P format are defined to represent frequency indices representative of symbols in the critical band pairs (block **812**). For example, referring to FIG. **10**, the hexadecimal number F1E4B0F is in base **16**, which matches P. The first digit of the hexadecimal number maps to a frequency component in the first critical band pair, the second digit to the second critical band pair, and so on. Each digit represents the frequency index that will be used to represent the symbol corresponding to the hexadecimal number F1E4B0F.

[0091] Using the first hexadecimal number as an example of mapping to a particular frequency index, the decimal value of Fh is 15. Because position four of each critical band pair is reserved for non-data information, the value of any hexadecimal digit greater than four is incremented by the value of one decimal. Thus, the **15** becomes a **16**. The **16** is thus designated (as shown with the asterisk in FIG. **10**) as being the code frequency component in the first critical band pair to represent the symbol corresponding to the hexadecimal number F1E4B0F. Though not shown in FIG. **10**, the index **1** position (e.g., the second position from the far left in the critical band **7** would be used to represent the hexadecimal number F1E4B0F.

[0092] After assigning the representative code frequencies (block **812**), the numbers are filled into a LUT (block **814**).

[0093] Of course, the systems and processes described in conjunction with FIGS. **8-10** are only examples that may be used to generate LUTs having desired properties in conjunction the encoding and decoding systems described herein. Other configurations and processes may be used.

Audio Decoding

[0094] In general, the decoder **116** detects a code signal that was inserted into received audio to form encoded audio at the encoder **102**. That is, the decoder **116** looks for a pattern of emphasis in code frequencies it processes. Once the decoder **116** has determined which of the code frequencies have been emphasized, the decoder **116** determines, based on the emphasized code frequencies, the symbol present within the encoded audio. The decoder **116** may record the symbols, or may decode those symbols into the codes that were provided to the encoder **102** for insertion into the audio.

[0095] In one implementation, the example decoder **116** of FIG. **11** may be implemented using, for example, a digital signal processor programmed with instructions to implement components of the decoder **116**. Of course, any other implementation of the example decoder **116** is possible. For example, the decoder **116** may be implemented using one or more processors, programmable logic devices, or any suitable combination of hardware, software, and firmware.

[0096] As shown in FIG. **11**, an example decoder **116** includes a sampler **1102**, which may be implemented using an analog to digital converter (A/D) or any other suitable technology, to which encoded audio is provided in analog format. As shown in FIG. **1**, the encoded audio may be provided by a wired or wireless connection to the receiver **110**. The sampler **1102** samples the encoded audio at, for example, a sampling frequency of 8 KHz. Of course, other sampling frequencies may be advantageously selected in order to increase resolution or reduce the computational load at the time of decoding. At a sampling frequency of 8 KHz, the Nyquist frequency is 4 KHz and, therefore, all of the embedded code signal is preserved because its spectral frequencies are lower than the Nyquist frequency. The 9216-sample FFT long block length at 48 KHz sampling rate is reduced to 1536 samples at 8 KHz sampling rate. However even at this modified DFT block size, the code frequency indices are identical to the original encoding frequencies and range from 180 to 656.

[0097] The samples from the sampler **1102** are provided to a stacker **1104**. In general, the stacker **1104** accentuates the code signal in the audio signal information by taking advantage of the fact that messages are repeated or substantially repeated (e.g., only the least significant bits are changed) for a period of time. For example, 42 bits (**226** of FIG. **2**) of the 49 bits (**226** and **224**) of the previously described example message of FIG. **2** remain constant for 64 seconds (32 2-second message intervals) when the 42 bits of data **226** in the message include a station identifier and a coarse time stamp which increments once every 64 seconds. The variable data in the last 7 bit group **232** represents time increments in seconds and, thus, varies from message to message. The example stacker **1104** aggregates multiple blocks of audio signal information to accentuate the code signal in the audio signal information. In an example implementation, the stacker **1104** comprises a buffer to store multiple samples of audio information. For example, if a complete message is embedded in two seconds of audio, the buffer may be twelve seconds long to store six messages. The example stacker **1104** additionally comprises an adder to sum the audio signal information associated with the six messages and a divider to divide the sum by the number of repeated messages selected (e.g., six).

[0098] By way of example, a watermarked signal y (t) can be represented by the sum of the host signal x(t) and watermark w (t):

[00001] $y(t) = x(t) + w(t)$

[0099] In the time domain, watermarks may repeat after a known period **7**:

[00002] $w(t) = w(t - T)$

[0100] According to an example stacking method, the input signal y (t) is replaced by a stacked signal S (t):

[00003] $S(t) = \frac{y(t) + y(t - T) + \text{ .Math. } + y(t - (n - 1)T)}{n}$

[0101] In the stacked signal S (t), the contribution of the host signal decreases because the values of samples x(t), x(t-T), . . . , x(t-nT) are independent if the period T is sufficiently large. At the same time, the contribution of the watermarks being made of, for example, in-phase sinusoids, is enhanced.

[00004] $S(t) = \frac{x(t) + x(t - T) + \text{ .Math. } + x(t - (n - 1)T)}{n} + w(t)$

[0102] Assuming x(t), x(t-T), . . . , x(t-nT) are independent random variables drawn from the same distribution X with zero mean E [X]=0 we obtain:

[00005] $\lim_{n \text{ .fwdarw. } \infty} E[\frac{x(t) + x(t - T) + \text{ .Math. } + x(t - (n - 1)T)}{n}]$ .fwdarw. 0, and

$\text{Var}[\frac{x(t) + x(t - T) + \text{ .Math. } + x(t - (n - 1)T)}{n}] = \frac{\text{Var}(X)}{n}$

[0103] Accordingly, the underlying host signal contributions x(t), . . . , x(t-nT) will effectively be canceling each other while the watermark is unchanged allowing the watermark to be more easily detected.
[0104] In the illustrated example, the power of the resulting signal decreases linearly with the number of stacked signals n. Therefore, averaging over independent portions of the host signal can reduce the effects of interference. The watermark is not affected because it will always be added in-phase.
[0105] An example process for implementing the stacker **1104** is described in conjunction with FIG. **12**.
[0106] The decoder **116** may additionally include a stacker controller **1106** to control the operation of the stacker **1104**. The example stacker controller **1106** receives a signal indicating whether the stacker **1104** should be enabled or disabled. For example, the stacker controller **1106** may receive the received audio signal and may determine if the signal includes significant noise that will distort the signal and, in response to the determination, cause the stacker to be enabled. In another implementation, the stacker controller **1106** may receive a signal from a switch that can be manually controlled to enable or disable the stacker **1104** based on the placement of the decoder **116**. For example, when the decoder **116** is wired to the receiver **110** or the microphone **120** is placed in close proximity to the speaker **114**, the stacker controller **1106** may disable the stacker **1104** because stacking will not be needed and will cause corruption of rapidly changing data in each message (e.g., the least significant bits of a timestamp). Alternatively, when the decoder **116** is located at a distance from the speaker **114** or in another environment where significant interference may be expected, the stacker **1104** may be enabled by the stacker controller **1106**. Of course, any type of desired control may be applied by the stacker controller **1106**.
[0107] The output of the stacker **1104** is provided to a time to frequency domain converter **1108**. The time to frequency domain converter **1108** may be implemented using a discrete Fourier transformation (DFT), or any other suitable technique to convert time-based information into frequency-based information. In one example, the time to frequency domain converter **1108** may be implemented using a sliding long block fast Fourier transform (FFT) in which a spectrum of the code frequencies of interest is calculated each time eight new samples are provided to the example time to time to frequency domain converter **1108**. In one example, the time to frequency domain converter **1108** uses 1,536 samples of the encoded audio and determines a spectrum therefrom using 192 slides of eight samples each. The resolution of the spectrum produced by the time to frequency domain converter **1108** increases as the number of samples used to generate the spectrum is increased. Thus, the number of samples processed by the time to frequency domain converter **1108** should match the resolution used to select the indices in the tables of FIGS. **3**-**5**.
[0108] The spectrum produced by the time to frequency domain converter **1108** passes to a critical band normalizer **1110**, which normalizes the spectrum in each of the critical bands. In other words, the frequency with the greatest amplitude in each critical band is set to one and all other frequencies within each of the critical bands are normalized accordingly. For example, if critical band one includes frequencies having amplitudes of 112, 56, 56, 56, 56, 56, and 56, the critical band normalizer would adjust the frequencies to be 1, 0.5, 0.5, 0.5, 0.5, 0.5, and 0.5. Of course, any desired maximum value may be used in place of one for the normalization. The critical band normalizer **1110** outputs the normalized score for each of the frequencies of the interest.
[0109] The spectrum of scores produced by the critical band normalizer **1110** is passed to the symbol scorer **1112**, which calculates a total score for each of the possible symbols in the active symbol table. In an example implementation, the symbol scorer **1112** iterates through each symbol in the symbol table and sums the normalized score from the critical band normalizer **1110** for each of the frequencies of interest for the particular symbol to generate a score for the particular symbol. The symbol scorer **1112** outputs a score for each of the symbols to the max score selector **1114**, which selects the symbol with the greatest score and outputs the symbol and the score.

[0110] The identified symbol and score from the max score selector **1114** are passed to the comparator **1116**, which compares the score to a threshold. When the score exceeds the threshold, the comparator **1116** outputs the received symbol. When the score does not exceed the threshold, the comparator **1116** outputs an error indication. For example, the comparator **1116** may output a symbol indicating an error (e.g., a symbol not included in the active symbol table) when the score does not exceed the threshold. Accordingly, when a message has been corrupted such that a great enough score (i.e., a score that does not exceed the threshold) is not calculated for a symbol, an error indication is provided. In an example implementation, error indications may be provided to the stacker controller **1106** to cause the stacker **1104** to be enabled when a threshold number of errors are identified (e.g., number of errors over a period of time, number of consecutive errors, etc.).

[0111] The identified symbol or error from the comparator **1116** is passed to the circular buffers **1118** and the pre-existing code flag circular buffers **1120**. An example implementation of the standard buffers **1118** is described in conjunction with FIG. **15**. The example circular buffers **1118** comprise one circular buffer for each slide of the time domain to frequency domain converter **1108** (e.g., **192** buffers). Each circular buffer of the circular buffers **1118** includes one storage location for the synchronize symbol and each of the symbol blocks in a message (e.g., eight block messages would be stored in eight location circular buffers) so that an entire message can be stored in each circular buffer. Accordingly, as the audio samples are processed by the time domain to frequency domain converter **1108**, the identified symbols are stored in the same location of each circular buffer until that location in each circular buffer has been filled. Then, symbols are stored in the next location in each circular buffer. In addition to storing symbols, the circular buffers **1118** may additionally include a location in each circular buffer to store a sample index indicating the sample in the audio signal that was received that resulted in the identified symbol.

[0112] The example pre-existing code flag circular buffers **1120** are implemented in the same manner as the circular buffers **1118**, except the pre-existing code flag circular buffers **1120** include one location for the pre-existing code flag synchronize symbol and one location for each symbols in the pre-existing code flag message (e.g., an pre-existing code flag synchronize that includes one message symbol would be stored in two location circular buffers). The pre-existing code flag circular buffers **1120** are populated at the same time and in the same manner as the circular buffers **1118**.

[0113] The example message identifier **1122** analyzes the circular buffers **1118** and the pre-existing code flag circular buffers **1120** for a synchronize symbol. For example, the message identifier **1122** searches for a synchronize symbol in the circular buffers **1118** and an pre-existing code flag synchronize symbol in the pre-existing code flag circular buffers **1120**. When a synchronize symbol is identified, the symbols following the synchronize symbol (e.g., seven symbols after a synchronize symbol in the circular buffers **1118** or one symbol after an pre-existing code flag synchronize symbol in the pre-existing code flag circular buffers **1120**) are output by the message identifier **1122**. In addition, the sample index identifying the last audio signal sample processed is output.

[0114] The message symbols and the sample index output by the message identifier **1122** are passed to the validator **1124**, which validates each message. The validator **1124** includes a filter stack that stores several consecutively received messages. Because messages are repeated (e.g., every 2 seconds or 16,000 samples at 8 KHz), each message is compared with other messages in the filter stack that are separated by approximately the number of audio samples in a single message to determine if a match exists. If a match or substantial match exists, both messages are validated. If a message cannot be identified, it is determined that the message is an error and is not emitted from the validator **1124**. In cases where messages might be affected by noise interference, messages might be considered a match when a subset of symbols in a message match the same subset in another already validated message. For example, if four of seven symbols in a message

match the same four symbols in another message that has already been validated, the message can be identified as partially validated. Then, a sequence of the repeated messages can be observed to identify the non-matching symbols in the partially validated message.
[0115] The validated messages from the validator **1124** are passed to the symbol to bit converter **1126**, which translates each symbol to the corresponding data bits of the message using the active symbol table.
[0116] An example decoding process **1200** is shown in FIG. **12**. The example process **1200** may be carried out by the example decoder **116** shown in FIG. **11**, or by any other suitable decoder. The example process **1200** begins by sampling audio (block **1202**). The audio may be obtained via an audio sensor, a hardwired connection, via an audio file, or through any other suitable technique. As explained above the sampling may be carried out at 8,000 Hz, or any other suitable frequency.
[0117] As each sample is obtained, the sample is aggregated by a stacker such as the example stacker **1104** of FIG. **11** (block **1204**). An example process for performing the stacking is described in conjunction with FIG. **13**.
[0118] The new stacked audio samples from the stacker process **1204** are inserted into a buffer and the oldest audio samples are removed (block **1206**). As each sample is obtained, a sliding time to frequency conversion is performed on a collection of samples including numerous older samples and the newly added sample obtained at blocks **1202** and **1204** (block **1208**). In one example, a sliding FFT may be used to process streaming input samples including 9215 old samples and the one newly added sample. In one example, the FFT using 9216 samples results in a spectrum having a resolution of 5.2 Hz.
[0119] After the spectrum is obtained through the time to frequency conversion (block **1208**), the transmitted symbol is determined (block **1210**). An example process for determining the transmitted symbol is described in conjunction with FIG. **14**.
[0120] After the transmitted message is identified (block **1210**), buffer post processing is performed to identify a synchronize symbol and corresponding message symbols (block **1212**). An example process for performing post-processing is described in conjunction with FIG. **15**.
[0121] After post processing is performed to identify a transmitted message (block **1212**), message validation is performed to verify the validity of the message (block **1214**). An example process for performing the message validation is described in conjunction with FIG. **18**.
[0122] After a message has been validated (block **1214**), the message is converted from symbols to bits using the active symbol table (block **1216**). Control then returns to block **1106** to process the next set of samples.
[0123] FIG. **13** illustrates an example process for stacking audio signal samples to accentuate an encoded code signal to implement the stack audio process **1204** of FIG. **12**. The example process may be carried out by the stacker **1104** and the stacker controller **1106** of FIG. **11**. The example process begins by determining if the stacker control is enabled (block **1302**). When the stacker control is not enabled, no stacking is to occur and the process of FIG. **13** ends and control returns to block **1206** of FIG. **12** to process the audio signal samples unstacked.
[0124] When the stacker control is enabled, newly received audio signal samples are pushed into a buffer and the oldest samples are pushed out (block **1304**). The buffer stores a plurality of samples. For example, when a particular message is repeatedly encoded in an audio signal every two seconds and the encoded audio is sampled at 8 KHz, each message will repeat every 16,000 samples so that buffer will store some multiple of 16,000 samples (e.g., the buffer may store six messages with a 96,000 sample buffer). Then, the stacker **1108** selects substantially equal blocks of samples in the buffer (block **1306**). The substantially equal blocks of samples are then summed (block **1308**). For example, sample one is added to samples 16,001, 32,001, 48,001, 64,001, and 80,001, sample two is added to samples 16,002, 32,002, 48,002, 64,002, 80,002, sample 16,000 is added to samples 32,000, 48,000, 64,000, 80,000, and 96,000.
[0125] After the audio signal samples in the buffer are added, the resulting sequence is divided by

the number of blocks selected (e.g., six blocks) to calculate an average sequence of samples (e.g., 16,000 averaged samples) (block **1310**). The resulting average sequence of samples is output by the stacker (block **1312**). The process of FIG. **13** then ends and control returns to block **1206** of FIG. **12**.

[0126] FIG. **14** illustrates an example process for implementing the symbol determination process **1210** after the received audio signal has been converted to the frequency domain. The example process of FIG. **14** may be performed by the decoder **116** of FIGS. **1** and **11**. The example process of FIG. **14** begins by normalizing the code frequencies in each of the critical bands (block **1402**). For example, the code frequencies may be normalized so that the frequency with the greatest amplitude is set to one and all other frequencies in that critical band are adjusted accordingly. In the example decoder **116** of FIG. **11**, the normalization is performed by the critical band normalizer **1110**.

[0127] After the frequencies of interest have been normalized (block **1402**). The example symbol scorer **1112** selects the appropriate symbol table based on the previously determined synchronization table (block **1404**). For example, a system may include two symbol tables: one table for a normal synchronization and one table for an pre-existing code flag synchronization. Alternatively, the system may include a single symbol table or may include multiple synchronization tables that may be identified by synchronization symbols (e.g., cross-table synchronization symbols). The symbol scorer **1112** then computes a symbol score for each symbol in the selected symbol table (block **1406**). For example, the symbol scorer **1112** may iterate across each symbol in the symbol table and add the normalized scores for each of the frequencies of interest for the symbol to compute a symbol score.

[0128] After each symbol is scored (block **1406**), the example max score selector **1114** selects the symbol with the greatest score (block **1408**). The example comparator **1116** then determines if the score for the selected symbol exceeds a maximum score threshold (block **1410**). When the score does not exceed the maximum score threshold, an error indication is stored in the circular buffers (e.g., the circular buffers **1118** and the pre-existing code flag circular buffers **1120**) (block **1412**). The process of FIG. **14** then completes and control returns to block **1212** of FIG. **12**.

[0129] When the score exceeds the maximum score threshold (block **1410**), the identified symbol is stored in the circular buffers (e.g., the circular buffers **1118** and the pre-existing code flag circular buffers **1120**) (block **1414**). The process of FIG. **14** then completes and control returns to block **1212** of FIG. **12**.

[0130] FIG. **15** illustrates an example process for implementing the buffer post processing **1212** of FIG. **12**. The example process of FIG. **15** begins when the message identifier **1122** of FIG. **11** searches the circular buffers **1118** and the circular buffers **1120** for a synchronization indication (block **1502**).

[0131] For example, FIG. **16** illustrates an example implementation of circular buffers **1118** and FIG. **17** illustrates an example implementation of pre-existing code flag circular buffers **1120**. In the illustrated example of FIG. **16**, the last location in the circular buffers to have been filled is location three as noted by the arrow. Accordingly, the sample index indicates the location in the audio signal samples that resulted in the symbols stored in location three. Because the line corresponding to sliding index **37** is a circular buffer, the consecutively identified symbols are 128, 57, 22, 111, 37, 23, 47, and 0. Because **128** in the illustrated example is a synchronize symbol, the message can be identified as the symbols following the synchronize symbol. The message identifier **1122** would wait until 7 symbols have been located following the identification of the synchronization symbol at sliding index **37**.

[0132] The pre-existing code flag circular buffers **1120** of FIG. **17** include two locations for each circular buffer because the pre-existing code flag message of the illustrated example comprises one pre-existing code flag synchronize symbol (e.g., symbol **254**) followed by a single message symbol. According to the illustrated example of FIG. **2**, the pre-existing code flag data block **230** is

embedded in two long blocks immediately following the 7 bit timestamp long block **228**. Accordingly, because there are two long blocks for the pre-existing code flag data and each long block of the illustrated example is 1,536 samples at a sampling rate of 8 KHz, the pre-existing code flag data symbol will be identified in the pre-existing code flag circular buffers **3072** samples after the original message. In the illustrated example FIG. **17**, sliding index **37** corresponds to sample index **38744**, which is 3072 samples later than sliding index **37** of FIG. **16** (sample index **35672**). Accordingly, the pre-existing code flag data symbol **68** can be determined to correspond to the message in sliding index **37** of FIG. **16**, indicating that the message in sliding index **37** of FIG. **16** identifies an original encoded message (e.g., identifies an original broadcaster of audio) and the sliding index **37** identifies an pre-existing code flag message (e.g., identifies a re-broadcaster of audio).

[0133] Returning to FIG. **12**, after a synchronize or pre-existing code flag synchronize symbol is detected, messages in the circular buffers **1118** or the pre-existing code flag circular buffers **1120** are condensed to eliminate redundancy in the messages. For example, as illustrated in FIG. **16**, due to the sliding time domain to frequency domain conversion and duration of encoding for each message, messages are identified in audio data for a period of time (sliding indexes **37**-**39** contain the same message). The identical messages in consecutive sliding indexes can be condensed into a single message because they are representative of only one encoded message. Alternatively, condensing may be eliminated and all messages may be output when desired. The message identifier **1122** then stores the condensed messages in a filter stack associated with the validator **1124** (block **1506**). The process of FIG. **15** then ends and control returns to block **1214** of FIG. **12**.

[0134] FIG. **18** illustrates an example process to implement the message validation process **1214** of FIG. **12**. The example process of FIG. **12** may be performed by the validator **1124** of FIG. **11**. The example process of FIG. **18** begins when the validator **1124** reads the top message in the filter stack (block **1802**).

[0135] For example, FIG. **19** illustrates an example implementation of a filter stack. The example filter stack includes a message index, seven symbol locations for each message index, a sample index identification, and a validation flag for each message index. Each message is added at message index M**7** and a message at location MO is the top message that is read in block **1802** of FIG. **18**. Due to sampling rate variation and variation of the message boundary within a message identification, it is expected that messages will be separated by samples indexes of multiples of approximately 16,000 samples when messages are repeated every 16,000 samples.

[0136] Returning to FIG. **19**, after the top message in the filter stack is selected (block **1802**), the validator **1124** determines if the validation flag indicates that the message has been previously validated (block **1804**). For example, FIG. **19** indicates that message MO has been validated. When the message has been previously validated, the validator **1124** outputs the message (block **1812**) and control proceeds to block **1816**.

[0137] When the message has not been previously validated (block **1804**), the validator **1124** determines if there is another suitably matching message in the filter stack (block **1806**). A message may be suitably matching when it is identical to another message, when a threshold number of message symbols match another message (e.g., four of the seven symbols), or when any other error determination indicates that two messages are similar enough to speculate that they are the same. According to the illustrated example, messages can only be partially validated with another message that has already been validated. When a suitable match is not identified, control proceeds to block **1814**.

[0138] When a suitable match is identified, the validator **1124** determines if a time duration (e.g., in samples) between identical messages is proper (block **1808**). For example, when messages are repeated every 16,000 samples, it is determined if the separation between two suitably matching messages is approximately a multiple of 16,000 samples. When the time duration is not proper, control proceeds to block **1814**.

[0139] When the time duration is proper (block **1808**), the validator **1124** validates both messages by setting the validation flag for each of the messages (block **1810**). When the message has been validated completely (e.g., an exact match) the flag may indicate that the message is fully validated (e.g., the message validated in FIG. **19**). When the message has only been partially validated (e.g., only four of seven symbols matched), the message is marked as partially validated (e.g., the message partially validated in FIG. **19**). The validator **1124** then outputs the top message (block **1812**) and control proceeds to block **1816**.
[0140] When it is determined that there is not a suitable match for the top message (block **1806**) or that the time duration between a suitable match(es) is not proper (block **1808**), the top message is not validated (block **1814**). Messages that are not validated are not output from the validator **1124**.
[0141] After determining not to validate a message (blocks **1806**, **1808**, and **1814**) or outputting the top message (block **1812**), the validator **1816** pops the filter stack to remove the top message from the filter stack. Control then returns to block **1802** to process the next message at the top of the filter stack.
[0142] While example manners of implementing any or all of the example encoder **102** and the example decoder **116** have been illustrated and described above one or more of the data structures, elements, processes and/or devices illustrated in the drawings and described above may be combined, divided, re-arranged, omitted, eliminated and/or implemented in any other way. Further, the example encoder **102** and example decoder **116** may be implemented by hardware, software, firmware and/or any combination of hardware, software and/or firmware. Thus, for example, the example encoder **102** and the example decoder **116** could be implemented by one or more circuit(s), programmable processor(s), application specific integrated circuit(s) (ASIC(s)), programmable logic device(s) (PLD(s)) and/or field programmable logic device(s) (FPLD(s)), etc. For example, the decoder **116** may be implemented using software on a platform device, such as a mobile telephone. If any of the appended claims is read to cover a purely software implementation, at least one of the prior code detector **204**, the example message generator **210**, the symbol selector **212**, the code frequency selector **214**, the synthesizer **216**, the inverse FFT **218**, the mixer **220**, the overlapping short block maker **240**, the masking evaluator **242**, the critical band pair definer **702**, the frequency definer **704**, the number generator **706**, the redundancy reducer **708**, the excess reducer **710**, the code frequency definer **712**, the LUT filler **714**, the sampler **1102**, the stacker **1104**, the stacker control **1106**, the time domain to frequency domain converter **1108**, the critical band normalize **1110**, the symbol scorer **1112**, the max score selector **1114**, the comparator **1116**, the circular buffers **1118**, the pre-existing code flag circular buffers **1120**, the message identifier **1122**, the validator **1124**, and the symbol to bit converter **1126** are hereby expressly defined to include a tangible medium such as a memory, DVD, CD, etc. Further still, the example encoder **102** and the example decoder **116** may include data structures, elements, processes and/or devices instead of, or in addition to, those illustrated in the drawings and described above, and/or may include more than one of any or all of the illustrated data structures, elements, processes and/or devices.
[0143] FIG. **20** is a schematic diagram of an example processor platform **2000** that may be used and/or programmed to implement any or all of the example encoder **102** and the decoder **116**, and/or any other component described herein. For example, the processor platform **2000** can be implemented by one or more general purpose processors, processor cores, microcontrollers, etc. Additionally, the processor platform **2000** be implemented as a part of a device having other functionality. For example, the processor platform **2000** may be implemented using processing power provided in a mobile telephone, or any other handheld device.
[0144] The processor platform **2000** of the example of FIG. **20** includes at least one general purpose programmable processor **2005**. The processor **2005** executes coded instructions **2010** and/or **2012** present in main memory of the processor **2005** (e.g., within a RAM **2015** and/or a ROM **2020**). The processor **2005** may be any type of processing unit, such as a processor core, a

processor and/or a microcontroller. The processor **2005** may execute, among other things, example machine accessible instructions implementing the processes described herein. The processor **2005** is in communication with the main memory (including a ROM **2020** and/or the RAM **2015**) via a bus **2025**. The RAM **2015** may be implemented by DRAM, SDRAM, and/or any other type of RAM device, and ROM may be implemented by flash memory and/or any other desired type of memory device. Access to the memory **2015** and **2020** may be controlled by a memory controller (not shown).

[0145] The processor platform **2000** also includes an interface circuit **2030**. The interface circuit **2030** may be implemented by any type of interface standard, such as a USB interface, a Bluetooth interface, an external memory interface, serial port, general purpose input/output, etc. One or more input devices **2035** and one or more output devices **2040** are connected to the interface circuit **2030**.

[0146] Although certain example apparatus, methods, and articles of manufacture are described herein, other implementations are possible. The scope of coverage of this patent is not limited to the specific examples described herein. On the contrary, this patent covers all apparatus, methods, and articles of manufacture falling within the scope of the invention.

Claims

1. A computing system comprising a processor and a memory, the computing system configured to perform a set of acts comprising: sampling an audio signal at a given time; based on the sampling the audio signal at the given time, extracting a first message from the audio signal; comparing the first message with a second message stored in a filter stack, wherein the second message is extracted at a previous time; based on: a match between the first message and the second message, and a time difference between the previous time and the given time, validating the first message; and based on the validating the first message, outputting the first message for use in audience measurement crediting.

2. The computing system of claim 1, wherein the set of acts further comprises: sampling the audio signal at the previous time; based on the sampling the audio signal at the previous time, extracting the second message from the sample of the audio signal; and storing the second message in the filter stack.

3. The computing system of claim 1, wherein the set of acts further comprises: based on: the match between the first message and the second message, and the time difference between the previous time and the given time, validating the second message; and based on the validating of the second message, outputting the second message for use in audience measurement crediting.

4. The computing system of claim 1, wherein: the filter stack stores a validation flag for the second message, and prior to the validating the first message, the validation flag for the second message indicates that the second message has not been validated.

5. The computing system of claim 1, wherein outputting the first message for use in audience measurement crediting comprises outputting the first message to a bit converter for translating symbols of the first message to corresponding data bits of the first message using a symbol table.

6. The computing system of claim 1, wherein extracting the first message comprises: obtaining a first sequence of samples of the audio signal; obtaining a second sequence of samples of the audio signal; averaging samples of the first sequence of samples with respective samples of the second sequence of samples to generate an average sequence of samples; and extracting the first message from average sequence of samples.

7. The computing system of claim 1, wherein extracting the first message comprises: identifying emphasized code frequencies; and determining, based on the emphasized code frequencies, a symbol encoded in the audio signal.

8. A computer-implemented method comprising: sampling an audio signal at a given time; based on

the sampling the audio signal at the given time, extracting a first message from the audio signal; comparing the first message with a second message stored in a filter stack, wherein the second message is extracted at a previous time; based on: a match between the first message and the second message, and a time difference between the previous time and the given time, validating the first message; and based on the validating the first message, outputting the first message for use in audience measurement crediting.

9. The computer-implemented method of claim 8, further comprising: sampling the audio signal at the previous time; based on the sampling the audio signal at the previous time, extracting the second message from the sample of the audio signal; and storing the second message in the filter stack.

10. The computer-implemented method of claim 8, further comprising: based on: the match between the first message and the second message, and the time difference between the previous time and the given time, validating the second message; and based on the validating of the second message, outputting the second message for use in audience measurement crediting.

11. The computer-implemented method of claim 8, wherein outputting the first message for use in audience measurement crediting comprises outputting the first message to a bit converter for translating symbols of the first message to corresponding data bits of the first message using a symbol table.

12. The computer-implemented method of claim 8, wherein: the filter stack stores a validation flag for the second message, and prior to the validating the first message, the validation flag for the second message indicates that the second message has not been validated.

13. The computer-implemented method of claim 8, wherein extracting the first message comprises: obtaining a first sequence of samples of the audio signal; obtaining a second sequence of samples of the audio signal; averaging samples of the first sequence of samples with respective samples of the second sequence of samples to generate an average sequence of samples; and extracting the first message from average sequence of samples.

14. The computer-implemented method of claim 8, wherein extracting the first message comprises: identifying emphasized code frequencies; and determining, based on the emphasized code frequencies, a symbol encoded in the audio signal.

15. A non-transitory computer-readable medium having stored therein instructions that, when executed by a computing system, cause the computing system to perform a set of acts comprising: sampling an audio signal at a given time; based on the sampling the audio signal at the given time, extracting a first message from the audio signal; comparing the first message with a second message stored in a filter stack, wherein the second message is extracted at a previous time; based on: a match between the first message and the second message, and a time difference between the previous time and the given time, validating the first message; and based on the validating the first message, outputting the first message for use in audience measurement crediting.

16. The non-transitory computer-readable medium of claim 15, wherein the set of acts further comprises: sampling the audio signal at the previous time; based on the sampling the audio signal at the previous time, extracting the second message from the sample of the audio signal; and storing the second message in the filter stack.

17. The non-transitory computer-readable medium of claim 15, wherein the set of acts further comprises: based on: the match between the first message and the second message, and the time difference between the previous time and the given time, validating the second message; and based on the validating of the second message, outputting the second message for use in audience measurement crediting.

18. The non-transitory computer-readable medium of claim 15, wherein: the filter stack stores a validation flag for the second message, and prior to the validating the first message, the validation flag for the second message indicates that the second message has not been validated.

19. The non-transitory computer-readable medium of claim 15, wherein outputting the first

message for use in audience measurement crediting comprises outputting the first message to a bit converter for translating symbols of the first message to corresponding data bits of the first message using a symbol table.

20. The non-transitory computer-readable medium of claim 15, wherein extracting the first message comprises: obtaining a first sequence of samples of the audio signal; obtaining a second sequence of samples of the audio signal; averaging samples of the first sequence of samples with respective samples of the second sequence of samples to generate an average sequence of samples; and extracting the first message from average sequence of samples.