# ADB2 Audio and Video Watermark Reader

1st Jan Beckschewe
*Technische Universität Berlin*
Berlin, Germany

2nd Alexander Schmitz
*Technische Universität Berlin*
Berlin, Germany

***Abstract*—Market analysis shows that many Hybrid Broadcast Broadband TV (*HbbTV*) capable consumer devices can't take advantage of the broadcast-related applications made available to them by broadcasters as the required metadata to discover these applications is often blocked before reaching the device. Therefore new specifications were created to give broadcasters the option to send an additional unique identifier in the form of a video and audio watermark to retrieve said metadata nonetheless. Creating a solution to decode these watermarks opens a new market for companies which offer HbbTV related services. This paper examines the encoding methods according to ETSI and ATSC standards and discusses the development of a real-time watermark decoder. Given a sample with adequate bitrate, our decoder was able to achieve a high decoding accuracy and reliably produced the unique identifier carried in the sample.**

## I. Introduction

Until now, receivers relied on signalling in the broadcast data to discover broadcast-related content in the internet. There are scenarios where this data, called Application Information Table (AIT), can not reach the HbbTV terminal. There are two scenarios mentioned in TS 103 464 [1] from which we will focus on the following case:

- A HbbTV capable TV set is connected via HDMI to a Set-Top-Box, connected to the DVB network. The AIT data is not carried through the HDMI and the end device has to rely on watermark data carried in the audio and video content to recover the AIT.

To reach the HbbTV application from the broadcast signal, one has to follow these steps:

- Extract the unique identifier in the form of a watermark from the broadcast data.
- From the unique identifier, resolve the AIT server through a DNS query.
- The AIT server will provide the channel specific AIT.
- The TV can then access the broadcast-related applications thanks to the AIT.

We will discuss the first step, where we extract the unique identifier from the watermarks in the broadcast signal.

## II. Related Work

The use of digital audio watermarks originated as a means to include information on copyright within the audio file [2] and later to include digital rights management information [3] to prevent unauthorised consumption of audible media. These methods were however rendered ineffective by modern high efficiency audio codecs. The first attempt of utilising autocorrelation difference functions was conducted in 2015 [4] and in 2019 the first successful attempt at embedding a watermark within the High Efficiency Advanced Audio Coding (HE-AAC) was carried out [5].

Similarly, in the video domain, the first use cases were intellectual property protection in forms that were both visible [6] and invisible [7] to the naked eye. Rudman et al. [8] describe a way to use video watermarks in real time to disable illegitimate online video streams in real time.

## III. Background

### A. Standards Organizations involved

The European Telecommunications Standards Institute (ETSI) is an European Standards Organization dealing with telecommunications, broadcasting and other communications networks and services. As such it provides standards complying to European regulations and legislation including the Technical Standard: TS 103 464 [1] which covers alternative discovery methods of HbbTV applications when AIT is not available through the broadcast network.

The Advanced Television Systems Committee, Inc. (ATSC) is an international, non-profit organization developing voluntary standards for digital television. The ETSI specification relies upon ATSC specifications for many of the key tasks, such as the lower data layers, i.e. the video and audio signal. The video watermark is contained in the pixels of a video signal and designed to withstand changes in format and compression. With the same reasoning, the audio watermark is contained in a human audible frequency band to survive compression and decompression or formatting of the original audio signal.

### B. Audio and Video physical Layer

To understand the decoding of the video watermark, we first have to understand the encoding standard a broadcaster has to use to embed their metadata in the video signal. The watermarking encoding uses manipulation of the luma values in the first row of pixels of each frame in the video stream. Each row carries 240 symbols independent of the resolution. For example a horizontal resolution of 1440 pixels results in 6 pixels per symbol. If the number of pixels in a row divided by 240 does not result in an integer number, the divided pixels luma value is distributed to the adjacent symbols using the fractional spacial contribution of the symbols to the pixel. An example for such a case is visualized in figure 1.

The broadcaster can choose between two systems for the video watermark encoding:

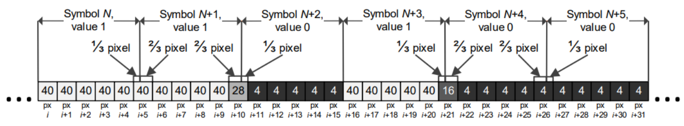
Symbol N, value 1
Symbol N+1, value 1
Symbol N+2, value 0
Symbol N+3, value 1
Symbol N+4, value 0
Symbol N+5, value 0
⅓ pixel
⅔ pixel
⅔ pixel
⅓ pixel
⅓ pixel
⅔ pixel
⅔ pixel
⅓ pixel
40 40 40 40 40 40 40 40 40 40 28 4 4 4 4 4 40 40 40 40 40 16 4 4 4 4 4 4 4 4 4 4
px i, px i+1, px i+2, px i+3, px i+4, px i+5, px i+6, px i+7, px i+8, px i+9, px i+10, px i+11, px i+12, px i+13, px i+14, px i+15, px i+16, px i+17, px i+18, px i+19, px i+20, px i+21, px i+22, px i+23, px i+24, px i+25, px i+26, px i+27, px i+28, px i+29, px i+30, px i+31

Fig. 1. Luma value calculation example. [9]

- The 1X system where each symbol corresponds to one bit.
- The 2X system where each symbol corresponds to two bits.

Depending on the distribution path and the display of the video, the flickering luma values in the first row of pixels can be noticeable. The 2X system also uses the full range of luma values with the brightest being close to white and the darkest close to black. The 1X system is less noticeable as the values are darker and therefore broadcasters are discouraged to use the 2X system if they are unsure if the receiver can mask the watermark.

Analogue to the video signal, we will look at the steps taken in the embedding of a watermark, to understand the decoding process. In order to avoid the loss of the watermark through re-encoding of the audio signal, the watermark is embedded in an audible frequency range of the audio signal, which means that the altering of the signal has to be kept at a minimum in order not to disturb the listening experience. This is made even more challenging by newly developed codecs which try to contain as little redundancy as possible.

### C. The Protocol

The ATSC defines different terms for the data stored in the watermarks for different stages in the decoding process. The general term watermark, or the method of watermarking refers to the delivery of data through the uncompressed audio and video. A watermark according to the ATSC 3.0 is a *wm_message* but in the context of this paper is mostly called just watermark. Its presence can be detected by checking a part of the data stream for a defined *run_in_pattern* which is a message header defining the start of a *wm_message*. There are over 10 different types of *wm_messages* defined by the ATSC and identifiable by a *wm_message_id* which is part of the *wm_message*. The type also tells the decoder the syntax and semantics necessary to decode the data bytes carried in the message block. There are multiple intermediate layers, but the highest layer, which is common to both video and audio, contains the *vp1_payload*, which is the most crucial part of the watermark because it contains the string used for the querying to receive the AIT data.

### D. Libraries we use

OpenCV (Open Source Computer Vision Library) is an open-source library for real-time computer vision applications. Although its functionality focuses on computer vision applications, it comes with many features for real-time image processing. As OpenCV is designed for real-time applications, efficiency is another key feature.

Pandas is a Python library mostly used for data manipulation and analysis, but it can also serve in applications performing signal processing. We use it specifically because to our knowledge, it is the only Python library to feature a correlation function for rolling data[1].

## IV. Our Approach

We aim to support as many different formats, codecs and real-time protocols as possible. This is made viable by offloading the demuxing and decoding and related works to *FFmpeg* [10], enabling us to work on the uncompressed RGB pixel values and on the uncompressed audio samples respectively. We support both input from prerecorded files as well as input from real-time streams. In terms of real-time input, we have tested both a server broadcasting a looped version of a sample video, as well as capturing the real-time output of the computer screen.

Because the FFmpeg decoding runs in its own process, we needed a way to communicate with our Python process. The most common approach to solve inter-process communication in a scenario with more than one data stream, is using named pipes [11], also called FIFO special files. However, the drawback of this approach is that it does not work consistently on all platforms, most notably the implementation differs on Microsoft Windows. We therefore used the unorthodox approach of repurposing the standard error anonymous pipe known as *stderr* to send the raw video stream and the standard output anonymous pipe known as *stdout* to send the raw audio stream. This setup is visualized in figure 2

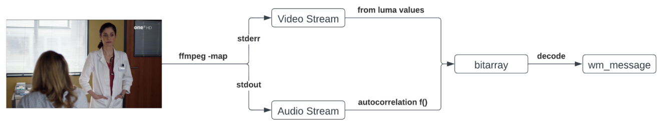

Fig. 2. Conceptual overview of decoding pipeline

### A. Reading Information from Video Stream

The first step is buffering a frame from the video stream on the stderr. For an 8-bit encoded video we can calculate the amount of bytes in a frame by multiplying the horizontal resolution with the vertical resolution times 3 (1 byte for each colour). As only the first row of pixels is important for the watermark decoding, we can discard the other rows to save resources in the following steps. The resulting array of pixels is then transformed to the $YC_BC_R$ (YCC) color space and from the resulting array we can further discard the $C_B$ and $C_R$ values to end up with a one dimensional array containing only the luma values of the first row of pixels. Figure 3 shows the luma value on the X-axis any the frame number over time on the Y-axis. It can be seen that the brightness value is not fully binary as we would hope but instead more

[1]https://pandas.pydata.org/docs/reference/api/pandas.core.window.rolling.Rolling.corr.html

continuous and in certain areas the values are significantly disturbed by compression artifacts to the point of making it nearly impossible to distinguish for both the machine as well as humans.

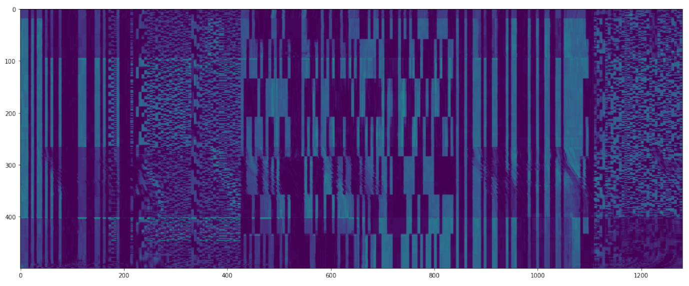

Fig. 3. Waterfall diagram of the video frame luma values

To resize the array from the size of the horizontal resolution of the original frame down to the 240 symbol length of the watermark message, we use linear interpolation. OpenCV comes with a *resize()* function which lets us specify the desired output size and interpolation method. With the linear interpolation method, we accommodate the above mentioned case, where dividing the number of pixels per row by 240 does not result in an integer solution. The resulting array $A$ with 240 symbols can then be mapped according to a threshold t to a bit array $B_V$ of equal size where:

$$B_{Vi} = \begin{cases} 0 & A_i \leq t \\ 1 & A_i > t \end{cases}$$

We found the algorithm suggested in A/335 [9] to return a bitarray with insufficient accuracy. Instead, we chose to adjust the threshold whether a pixel corresponds to a 0 or 1 bit after translating the whole first row into a bit array. If we detect the header of a watermark message, we know that the threshold is fitting. In less than 10% of the cases the header does not match the expected header, which means that some bits were flipped. In the next step we try other realistic thresholds until one is found which returns an expected header. If the header still does not fit, the whole message is discarded, as it is expected, that either the frame does not contain a watermark at all or if it does, the bits in the watermark are separated by an inconsistent threshold. In the case of an inconsistent threshold, i.e. some values larger than the threshold should be read as 0 while at the same time some values smaller than the threshold should be read as 1, there is no reliable way to identify the correct bitarray. However this is only a small issue because the same watermark is always repeated at least five times, leaving enough redundancy.

Table I shows the payload format for the video watermark for both the 1X and 2X system. After detecting the correct run_in_pattern (0xEB52) in the first 16 bits of the $B_V$ bit array, the watermark message block can be decoded. For the 1X system this block contains 136 bits, which can be decoded following the syntax from table II

| Syntax | No. of Bits | Format |
|---|---|---|
| watermark_payload(){ | | |
| **run_in_pattern** | 8 | 0xEB52 |
| for (i=0; i<N; i++){ | | |
| **wm_message_block()** | var | |
| } | | |
| for (i=0; i<M; i++){ | | |
| **zero_pad** | 8 | 0x00 |
| } | | |
| } | | |

TABLE I
BIT STREAM SYNTAX OF THE WATERMARK PAYLOAD. [12]

| Syntax | #Bits | Format |
|---|---|---|
| wm_message_block() { | | |
| **wm_message_id** | 8 | uimsbf |
| **wm_message_block_length** | 8 | uimsbf |
| **wm_message_version** | 4 | uimsbf |
| if ((wm_message_id & 0x80)==0) { | | |
| **fragment_number** | 2 | uimsbf |
| **last_fragment** | 2 | uimsbf |
| } else { | | |
| **reserved** | 4 | uimsbf |
| **fragment_number** | 8 | uimsbf |
| **last_fragment** | 8 | uimsbf |
| } | | |
| **wm_message_bytes()** | var | |
| if ((fragment_number == last_fragment) && (fragment_number != 0)) { | | |
| **message_CRC_32** | 32 | uimsbf |
| } | | |
| **CRC_32** | 32 | uimsbf |
| } | | |

TABLE II
BIT STREAM SYNTAX FOR THE WATERMARK MESSAGE BLOCK. [12]

## B. Reading Information from Audio Stream

The autocorrelation difference function (ADF) in A/334 [13] is given as:

$$R_d(i,t,\tau) = \int_t^{t+T/2} s'_i(u)s'_i(u-\tau)du - \int_{t+T/2}^{t+T} s'_i(u)s'_i(u-\tau)du$$

We faced significant hurdles in understanding this function but after we had overcome those by the long process of trial and error, we faced the next issue which was that our implementation was vastly too slow, approximately 10 times slower than real-time. In a first optimization step, using numpy instead of statsmodels, we reached 4 times real-time speed. However there was still potential for massive improvements because we realized that we were performing the calculation repeatedly over partly the same samples. In a further optimization step, we reached 60 times real-time speed for the ADF by utilizing the rolling correlation function in the pandas library. Figure 4 shows on the X-axis the frame number and on the Y-axis the amplitude of the signal after executing the ADF. Additional steps are however needed to decode the audio bitarray. The next step is finding out the timing of when a new cell of 1.5 seconds starts. This is achieved by searching for a specific pattern of bits within the bitstream, similar to the video run_in_pattern. We can then use our unified approach to decode both video and audio watermarks in the same manner.

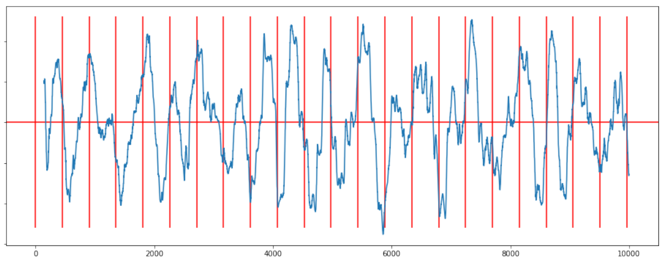

Fig. 4. Audio signal after ADF

## C. *Watermark messages*

The obtained bitarray from the video and audio stream can be decoded according to Table I. The payload of the watermark is nested within several layers of abstractions, each stripping a certain set of bits in order to enable e.g. the error correction. We are translating the watermark message into a number of Python dataclasses, many of them nested within each other for a object-oriented structure to avoid redundancy in the decoding process. Additionally, we check the form of the watermark message, for example the correct length and calculate the CRC32 Checksum (Cyclic Redundancy Check) of the *wm_message_block()* to detect any errors in the message block and if the message block is the last fragment, the CRC32 for the assembled watermark message can be checked.

## V. Evaluation/Discussion

We will first discuss results from the decoding of the video stream, then the decoding of the audio stream and finally the translation into watermark messages. The ATSC Standard A/335 [9] provided us with the necessary information to retrieve the bitarray from a watermark carrying image. However we found that the encoded luma values vary significantly between video samples and don't match the examples from the standard. This made decoding more challenging and confusing at the start. Considering the 1X system of encoding, a high amount of symbols had a luma value close to the threshold value. Figure 5 shows the relative distribution of the luma values in the 240 symbols per frame for 10,000 frames (only luma values $\leq$ 50).

The accuracy of the decoded watermark messages from the video stream gives us a good indicator on the quality of the decoding method. We calculate the accuracy by dividing the number of message blocks where the given CRC32 matches the calculated one by the number of bitarrays which carry the correct *run_in_pattern*. We found that there are two main factors influencing the accuracy. First, the distribution of luma values, where a larger gap between values mapped to 0 and those mapped to 1 is generally better. In other words, less values close to the threshold results in better accuracy. Second, the biggest impact on the accuracy is held by the bitrate at which the video is encoded. As figure 6 shows, a bitrate of 1 Mbit/s made it almost impossible to decode any video watermark and the accuracy dropped to less than 3%.

The accuracy of the decoded watermark messages from the audio stream is much more consistent than the accuracy for video which is due to the fact that there is less of a need for stronger compression in audio because it always contains vastly fewer bits than video. As an example, a combined audio/video stream encoded at 5 MBit/s usually uses the same audio encoding settings as a 1 Mbit/s combined stream. It should be noted that the audio accuracy data should be taken with caution because we have noticed that these messages often contain errors which were not caught because we did not succeed in utilizing the BCH codes. However it stands to reason that in the vast majority of cases, the original message would be fully recovered with no errors left, if BCH decoding would work. The BCH variant used in the specification can correct up to 13 bitflips.

Performance wise, we are reaching approximately four times real-time speed for the entire pipeline with decoding, splitting and video and audio decoding on reasonably powerful desktop machine, however performance will be significantly worse on embedded hardware like television sets.

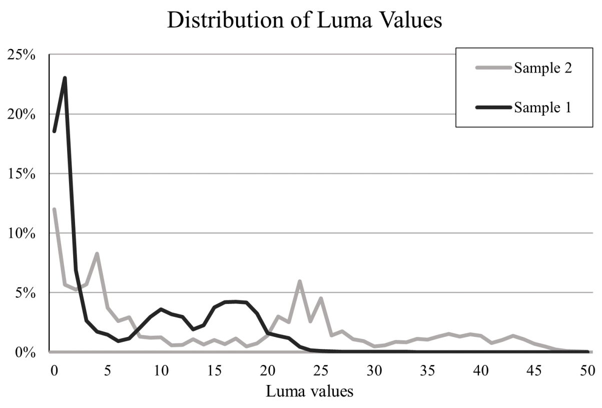

Fig. 5. Distribution of Luma Values of the 240 symbols after the linear interpolation (Sample size 10000 frames).

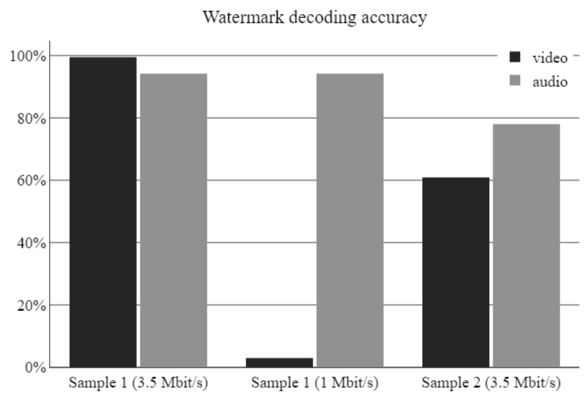

Fig. 6. Decoding accuracy for different audio/video samples.

## VI. Conclusion

We have successfully built a prototype that complies with the ETSI specification in the most important aspects to enable HbbTV using audio and video watermarks. We have also made our contribution freely available to everyone interested on GitHub[2]. This marks an important step in increasing the reach of this technology, which can improve the viewer's experience as well as generate additional revenue streams for broadcasters through advertisements.

### A. Future Works

The VP1 messages contain error-correcting codes based on BCH codes in order to correct errors in the received bitstream. Future works are needed to decode these, in order to eliminate the possibility of wrongly decoding audio VP1 payloads. A python library[3] exists for decoding BCH codes but we have found it to be inadequate for our use case. Additionally, we have only decoded a subset of the available message IDs, extended_vp1_message and vp1_message. For full standard compliance, the entire set of message IDs including dynamic_event_message and content_id_message are required. Even though we have only discussed the extraction of the watermark in this paper, we did peak into the issue of resolving the response we received, but the server we queried did not return the expected result, meaning more future work will be needed.

## References

[1] *Hybrid Broadcast Broadband TV; Application Discovery over Broadband*. Standard. European Telecommunications Standards Institute, Jan. 2022.

[2] Laurence Boney, Ahmed H Tewfik, and Khaled N Hamdy. "Digital watermarks for audio signals". In: *Proceedings of the third IEEE international conference on multimedia computing and systems*. IEEE. 1996, pp. 473–480.

[3] Patrick Wolf, Martin Steinebach, and Konstantin Diener. "Complementing DRM with digital watermarking: mark, search, retrieve". In: *Online Information Review* (2007).

[4] Habibur Muhaimin et al. "An efficient audio watermark by autocorrelation methods". In: *2015 International Conference on Electrical Engineering and Informatics (ICEEI)*. IEEE. 2015, pp. 606–611.

[5] Yiyu Hong and Jongweon Kim. "Autocorrelation modulation-based audio blind watermarking robust against high efficiency advanced audio coding". In: *Applied Sciences* 9.14 (2019), p. 2780.

[6] Jianhao Meng and Shih-Fu Chang. "Embedding visible video watermarks in the compressed domain". In: *Proceedings 1998 International Conference on Image Processing. ICIP98 (Cat. No. 98CB36269)*. Vol. 1. IEEE. 1998, pp. 474–477.

[7] Raymond B Wolfgang, Christine I Podilchuk, and Edward J Delp. "Perceptual watermarks for digital images and video". In: *Proceedings of the IEEE* 87.7 (1999), pp. 1108–1126.

[8] Ken Rudman et al. "Toward real-time detection of forensic watermarks to combat piracy by live streaming". In: *SMPTE 2014 Annual Technical Conference & Exhibition*. SMPTE. 2014, pp. 1–12.

[9] *Video Watermark Emission*. Standard. Advanced Television Systems Committee, Mar. 2022.

[10] Suramya Tomar. "Converting video formats with FFmpeg". In: *Linux Journal* 2006.146 (2006), p. 10.

[11] Andy Vaught. "Linux apprentice: Introduction to named pipes". In: *Linux Journal* 1997.41es (1997), 18–es.

[12] *Content Recovery in Redistribution Scenarios*. Standard. Advanced Television Systems Committee, Mar. 2022.

[13] *Audio Watermark Emission*. Standard. Advanced Television Systems Committee, Mar. 2022.

[2]https://github.com/adb2-watermark-reader/adb2-watermark-reader
[3]https://github.com/jkent/python-bchlib