

# MPEG-1/2 audio decoder using a System-on-Chip with a RISC-V processor

Henrique Alves Gonçalves  
henriquealvesg@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

December 2024

## Abstract

This work consists of developing a system for audio decoding according to the MPEG 1/2 compression standard. The format is widely used in the multimedia device industry and sound interfaces, typically in infrastructure dedicated to digital signal processing or digital systems and reference microcontrollers such as x86 and ARM processors. The study's primary objective was implementing the decoder in an embedded system with a central processing unit with RISC-V ISA. This type of simplified specification provides extensions created for different computing domains and is distinguished by adopting an open-source license. The system's structure is based on the IOB-SoC platform, implemented in the Verilog hardware description language. The software integrates the open-source library *LibMAD* into the system's base architecture. In the context of FPGA-supported prototyping, tests were carried out to estimate the computational execution time of the algorithm, revealing minimal usage of components in the device structure. The implementation result is sufficient to meet the objective of successful decompression and data integrity with the PicoRV and real-time processing with the DarkRV and VexRV processors.

**Keywords:** Open-source, MPEG, Audio decoder, System-on-Chip, RISC-V, Field-programmable gate array

## 1. Introduction

### 1.1. Context

In today's rapidly evolving technological landscape, there is a discernible shift towards hybrid approaches in computing system development from lower-end systems, such as microcontrollers, to more complex computing machines, for example, with machine learning or hardware acceleration dedicated modules. Such advancements transcend conventional boundaries, permeating diverse domains and driving transformative change across various use cases. This perpetual evolution necessitates agile methodologies and adaptable frameworks to accommodate the ever-changing demands of contemporary applications.

Embedded system design was once a matter of manually transposing an intention expressed using mnemonics into machine code (usually represented using hexadecimal or binary values) and storing those values in a persistent storage device readable by a microprocessor. This technology is integral to numerous modern devices, consisting of processing units, memory, and input/output peripherals. It is developed to perform dedicated functions within larger systems, materializing the convergence of software and hardware.

The system-on-chip paradigm embodies the composition of diverse intellectual property blocks into a cohesive design, culminating in a versatile platform that can orchestrate intricate functionalities for data processing and operating systems environments. Tailored to specific applications, it leverages targeted optimizations and to fulfil specialized requirements with precision and efficiency.

### 1.2. Motivation

With the development of processor design and semiconductor technology, RISC's digital signal processing capability has approached the DSP level. Therefore, it became a reality to implement MP3 decoding in real-time on a single RISC core [1]. Audio data compression techniques allow audio engineers to save memory space by targeting sensitive and perceptually irrelevant signal data and quantising the difference between a masking amplitude envelope and the actual sound level.

This thesis report defines the proposed software implementation for the MPEG-1/2 audio decoder and provides a detailed description of the system's integration and validation.

### 1.3. Definition of objectives and deliverables

In this project, a software implementation of the MPEG-1/2 audio layers I, II and III decoder uses a RISC-V scalar unit with dedicated modules for fixed-point multiply and vision instructions and a five-stage pipeline processor installed on an FPGA for design, simulation, and test purposes.

Custom and modular firmware, debugging tools, and project build automation are essential elements in the development process of embedded systems, ensuring efficient compilation, synthesis, and deployment.

For the current project, a software library is used for the decoding process, *LibMAD* (Underbit Technologies) [2]. This repository will be implemented with scalar and pipelined central processing units based on RISC-V instruction set architecture and evaluated to guarantee the correct operation of the software implemented in the context of the IOB-SoC.

## 2. Background

### 2.1. Fundamentals of signal processing

#### Sampling theorem

The conversion from a continuous-time signal,  $x(t)$ , to a discrete-time signal is achieved through a process known as sampling. This method involves the measurement of the amplitude of the signal at discrete intervals, typically uniform and periodic. The mathematical expression for the continuous-time sampled signal,  $x_s(t)$ , and the discrete-time representation,  $x[n]$ , can be written as:

$$x_s(t) = \sum_{n=-\infty}^{\infty} x(nT_s) \cdot \delta(t - nT_s) \quad (1)$$

$$x[n] = x(nT_s) \quad (2)$$

where  $T_s$  is the sampling period, the time interval between successive samples,  $\delta(t - nT_s)$  is the Dirac delta function, creating impulses at each sampling instant  $t = x(nT_s)$  and  $n$  is an integer representing the index of the sample. In the frequency domain, this multiplication corresponds to a convolution between the signal's spectrum  $X(f)$  and the Fourier Transform of the impulse train, which is another impulse train:

$$X_s(f) = X(f) * \left( f_s \cdot \sum_{k=-\infty}^{\infty} \delta(f - kf_s) \right) \quad (3)$$

This convolution results in replicas of the original spectrum  $X(f)$  shifted by multiples of the sampling frequency  $f_s$ .

The Nyquist-Shannon theorem provides a criterion for the minimum sampling rate that must be used to ensure that the continuous-time signal can be fully reconstructed from its samples without any information loss. Aliasing is a phenomenon that causes different signals to become indistinguishable

(or aliases of one another) when sampled. It occurs when the sampling frequency  $f_s$  is insufficient to capture the signal's frequency content.

Assuming the signal is band-limited to a maximum frequency  $B$  (i.e.,  $X(f) = 0$  for  $|f| > B$ ), the components in the frequency domain due to sampling will start at  $\pm f_s$  and extend from  $f = \pm f_s - B$  to  $f = \pm f_s + B$ . To prevent overlap between the original spectrum and the transformed signal, it is required:

$$f_s - B > B \implies f_s > 2B \quad (4)$$

This inequality ensures that the highest frequency component of the original signal does not interfere with the lowest frequency component of the adjacent replica, thereby preventing aliasing, which can be described by the folding of frequency components beyond the Nyquist frequency ( $f_{\text{Nyquist}} = f_s/2$ ) back into the range  $[0, f_{\text{Nyquist}}]$  [3].

#### Quantization and encoding

After sampling, the continuous amplitude values must be quantised into discrete levels. In PCM, the sampled value  $x[n]$  is approximated by the nearest value from a finite set of discrete amplitude levels. This step is necessary because digital systems can only represent a finite number of amplitude values due to their binary nature. It is the standard analogue audio conversion format used in digital systems. The difference between the actual and quantised values introduces quantisation error, a form of noise defined as the difference between the original sample and the quantised values. Following quantisation, each quantised sample  $x_q[n]$  is encoded into a binary sequence for digital representation. The encoding process assigns a unique binary code to each quantisation level. The number of bits  $b$  used in encoding determines the resolution and the number of quantisation levels  $L$ . The quantisation level set is determined by the quantisation resolution, which depends on the number of bits  $b$  used to represent each sample. The total number of discrete levels is given by  $L = 2^b$ .

#### Fourier transform and its applications

The Fourier Transform is a fundamental mathematical tool used to analyse and process signals in the frequency domain. It decomposes a time-domain signal into its constituent frequency components, providing a frequency spectrum that describes the signal's content [4]. For a continuous-time signal  $x(t)$ , the Fourier Transform  $X(f)$  expression is:

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft} dt \quad (5)$$

This transformation converts the time-domain signal into a continuous function of frequency, revealing the amplitude and phase of each frequency

component present in  $x(t)$ . For discrete-time signals obtained through sampling, the Discrete-time Fourier Transform (DTFT) provides a continuous frequency spectrum for a sampled signal  $x[n]$ . The DFT is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi}{N}kn}, \quad k = 0, 1, \dots, N-1 \quad (6)$$

where  $N$  is the number of samples and  $X[k]$  represents the frequency component at index  $k$ . In signal processing, the discrete signal transformation is often computed using the Fast Fourier Transform. This algorithm method takes advantage of the periodic and symmetric properties of the complex exponentials (roots of unity) in the DFT formula. It recursively splits the DFT of  $N$  points into two DFTs of  $N/2$  points, one for the even-indexed inputs and one for the odd-indexed inputs. The algorithm reduces the complexity of computing the DFT from  $O(N^2)$  to  $O(N \log_2(N))$ , making it much faster for larger  $N$ . This efficiency gain is crucial for applications requiring real-time or high-speed processing of large datasets, such as audio/ video processing and telecommunications.

## 2.2. MPEG standard - ISO/ IEC 11172 & 13818

The Moving Picture Experts Group was founded in 1988 by the International Standards Organization to develop coding standards for multimedia, including digital audio and video formats for compression and data transport. The first version of the codec standard, MPEG-1, identified by ISO/ IEC 11172-3, was completed in 1992 and describes three compression layers. A more advanced version, MPEG-2, identified by ISO/ IEC 13818, followed, further refining these standards and enabling improved efficiency and performance across various applications, including broadcasting and DVDs.

Both standards employ a layered approach, with layers presenting increasing complexity and higher compression rates, making the last layer the most efficient. This, combined with a broader, more flexible coding format than the anterior layers and a range of available configuration settings related to signal processing attributes, such as bit rate and sample frequencies, leads to more coverage regarding audio data content and its applications, one of the reasons why MP3 became extremely popular for digital audio files.

## 2.3. Compressed audio characteristics and applications

From communication to audiovisual consumer electronics and even with the emergence of the World Wide Web, along with the adoption of audio interfaces in consumer electronics of all application types, the decreased compressed data size becomes

a necessity to encode audio information as memory capacity on a chip is reduced. Maintaining this homogeneity and information integrity between the sender and receiver is essential for real-time audio and visual multimedia transmission to secure service quality at a reasonable bandwidth.

By integrating the mentioned functionalities using platform-based approaches, companies can develop dedicated hardware and/ or software to implement coding algorithms with multiple devices and environments. Low power consumption and the area associated with the hardware appliances are two main features of digital and logic systems design and deployment in small-scale designs. Audio decoding solutions as a software component on a system-on-chip offer a good relationship between application abstraction and hardware resources or required memory space.

Compressed audio standards have also paved the way for advancements in mobile and streaming applications. As streaming platforms became more popular, efficient compression algorithms that balance audio quality and data size became critical. MPEG-2's flexibility in handling lower bit rates makes it especially useful in limited bandwidth, such as mobile networks or online streaming services. Moreover, the compatibility of these standards with various playback devices—ranging from low-power embedded systems in portable devices to high-end multimedia systems illustrates their adaptability across different environments.

## 2.4. Decoding process

In the domain of data compression, a decoder is an element that transforms compressed data into the raw format that originated it as accurately as possible.

Audio decoding aims to accurately reconstruct the original audio signal from the compressed format, with lossless or lossy information integrity between processes. This process involves inverse operations of the encoding process, where the compressed data is expanded back into a form that closely approximates the original audio waveform, ideally with indistinguishable differences to the human ear.

The first step is error verification, which protects the integrity of information by examining it with a cyclic redundancy code [5]. The next step is to unpack the compressed data in the bitstream, from the header to the side and the primary information for the quantised samples that will be inversely quantised with the corresponding number of bits. Finally and analogously, a synthesis filter bank combines the resulting dequantized sub-band samples into a single or dual channel frequency domain to reproduce the output and recovered signal.

### 2.4.1 Bitstream unpacking

At first, the decoder should synchronize the bitstream at the beginning of the frame for decoding. Then, the side information of the MPEG audio basic channel is extracted. This information is utilized to dequantize and synthesize the subband samples. According to the channel information and dequantized subband samples, the multi-channel process can reconstruct another three channels (left, right and centre). After each channel's subband samples are ready, the synthesis filter bank can be started to generate the PCM samples.

The decoder performs entropy decoding, then reconstructs the quantized subband values and transforms subband values into a time-domain audio signal. Scale factors are used during encoding to adjust the quantisation step sizes for different frequency bands and are extracted for inverse quantisation. The decoder reads the bitstream and maps the variable-length codes to the quantised values using the code tables defined in the MPEG standard.

### 2.4.2 Frequency-to-time inverse mapping

Frequency-to-time inverse mapping converts a signal from the frequency to the time domain. An inverse transform, such as the Inverse Modified Discrete Cosine Transform (IMDCT), converts the frequency-domain coefficients back to time-domain samples.

In audio codecs like MP3, the IMDCT is essential in reconstructing audio signals after processing in the frequency domain. It operates by overlapping and adding subsequent blocks, which reduces interferences at the boundaries between blocks and ensures smooth audio reconstruction. The overlapping approach allows smoother transitions between successive audio frames, minimizing discontinuities and avoiding audible distortions. It provides a high-fidelity reconstruction, making it highly effective for perceptual audio coding.

After synchronizing with this sequence, the decoder determines other relevant information, such as the sampling rate of uncompressed data and the data rate of the compressed data stream. The decoder then uses this information to decode the other frames. The coding tables translate the variable length-coded symbols in the inverse quantisation phase. These symbols would have been generated by the encoder based on the frequency of occurrence of each symbol. The primary goal of Huffman coding is to minimize the average code length, thus reducing the overall bit rate of the encoded data.

### 2.4.3 Sub-band synthesis

In sub-band synthesis, the original signal is split into several frequency bands using a series of filters. Each band is then processed independently, including compression, modification, or other forms of manipulation. After processing, these bands are recombined and synthesised to reconstruct the signal. A synthesis filter bank reconstructs the audio signal by combining the time-domain samples from each subband to generate the final time-domain audio signal.

In the synthesis/ filterbank stage, two steps are performed. In the first step, groups of 32 subband samples provided by the initial decoding phase are converted to 64 entry arrays using a discrete cosine transform. There are 36 such groups, and each sample in a subband represents the amplitude for a particular frequency. At any point, the synthesis/ filterbank phase keeps a set of sixteen 64-entry arrays in a rotating window fashion. In the second step, the 64-entry arrays are windowed using a set of 512 coefficients to produce 32 PCM samples. Thus, the 36 groups per channel in a frame produce 1152 decoded audio samples. The signal is represented as a finite sequence of data points as a sum of cosine functions oscillating at different frequencies.

The synthesis filter bank in MPEG audio decoding employs a polyphase filterbank architecture to split and subsequently recombine the subbands smoothly. Polyphase filtering ensures seamless transitions between subbands, thus minimizing artifacts and maintaining audio quality during reconstruction. Relying on Quadrature Mirror Filters (QMF), specialized filters are designed to split the input signal into multiple subbands without introducing aliasing and maintain the signal's integrity during subband splitting. A QMF splits the input signal into two parts: one passes through a low-pass filter and the other through a high-pass filter. These two filters are "mirror images" of each other in frequency response, which helps ensure that the original signal can be perfectly reconstructed from the subbands. In MPEG decoding, this approach helps ensure that the split frequency components remain in phase and that energy is preserved across the bands.

## 2.5. Development technologies

### 2.5.1 Field Programmable Gate Array

As mentioned, field-programmable gate array devices encapsulate the physical computing system. It is a hardware interface made of digital components like a standard computer system but with the advantage of being composed of a dedicated fabric of reprogrammable logic that allows developers to create digital systems design, implementing com-

plex logic gate blueprints using Hardware Description Languages such as VHDL or Verilog and simultaneously leveraging the FPGA’s potential to achieve optimised computational throughput and low latency.

### 2.5.2 System-on-Chip

The SoC design approach has become a key enabler for advancements across the integrated computer chip industry. This has led to the emergence of platform-based design techniques, in which a more flexible, programmable or reconfigurable medium is reused across a set of designs within a specific application domain [6]. SoC designs, particularly in FPGA-based environments, now drive the development and adoption of industry-wide standards by offering a flexible and customizable architecture [7], making it

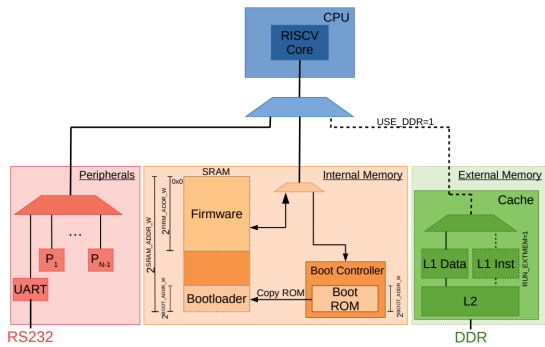


Figure 1: Block diagram of the system-on-chip hardware structure

### 2.5.3 The IOB-SoC project template

The system-on-chip developed by *IObundle*, *Lda* provides a RISC-V instruction set architecture-based platform appropriate for hardware and software development designated as IoB-SoC [8], a SoC template coded with Verilog hardware description language.

*IObundle* provides development tools such as interfaces and computing hardware to facilitate the necessary communications with the development boards. These resources are suited for testing and simulating the provided SoC architecture.

An overlay design, a virtual layer of architecture that is conceptually located between the user application and the underlying physical FPGA, is implemented to allow greater flexibility by abstracting hardware details, enabling easier reuse and development of the system for different applications [9]. This approach is further supported by a type of hardware abstraction layer which provides a uniform interface for software development independent of the underlying hardware. Using abstraction

facilitates portability across different hardware configurations, thus enhancing the versatility and scalability of the SoC. Therefore, this project template can be extended to implement customized processing systems with hardware and software solutions.

### 2.6. RISC-V Instruction Set Architecture

The RISC-V Foundation was created in 2015 by the Parallel Computing Laboratory (Par Lab) at the University of California at Berkeley. It is a free and open-source Instruction Set Architecture that provides a method to create intellectual property and a more accessible approach to the digital processing products and devices market. The system-level organization of a RISC-V hardware platform can vary significantly, ranging from single-core microcontrollers to large-scale clusters of many-core server nodes with shared memory. Even small systems-on-a-chip can adopt a hierarchical structure, incorporating multicomputer or multiprocessor configurations. This modular approach not only simplifies the development process but also enables secure isolation between subsystems, enhancing flexibility and security in various embedded and general-purpose applications [10].

This architecture type is composed of 32 general-purpose integer registers from x0 to x31, plus a program counter (instruction pointer), with an additional 32 floating-point registers for the implementation of the respective extension, except in the embedded subset variant, which has only 16 integer registers. These assembler mnemonics are based on register-address instruction roles, except for memory-access instructions that reside outside the scope of the integer registers.

Processor instructions include a set of arithmetic, logical, memory access, and control transfer instructions. Arithmetic and logical operations include addition, subtraction, shifts, and bitwise operations. The floating-point extension adds another set of 32 registers, which are crucial for efficiently handling arithmetic operations involving real numbers, particularly for applications such as scientific computation and multimedia processing.

The architecture is well-suited for embedded systems, providing a small footprint and a simple set of instructions that are easy to implement. The ISA’s minimalism and modular nature allow for flexible implementations, where different subsets and extensions can be chosen based on the application’s specific needs. In contrast, more general-purpose RISC-V implementations can include a broader range of extensions, enabling them to run complex operating systems and handle high-performance computing tasks.

### 3. Implementation

#### 3.1. Software architecture

The decoder in the system is implemented by combining the existing SoC firmware with the application programmable interface (API), which is natively implemented with the library. For that effect, the audio decompression source files collection is added as a *git submodule* present in the SoC's highest-level directory, making available the required functions, structures, variables and macro definitions in the upper layers of software of the system-on-chip. To streamline this process, developers use automated build tools and continuous integration techniques that ensure updates to the submodule do not disrupt existing system functionalities, thereby enhancing the robustness of the system's software infrastructure. Figure 2 shows a flow diagram representing the stages of the development/ verification tasks.

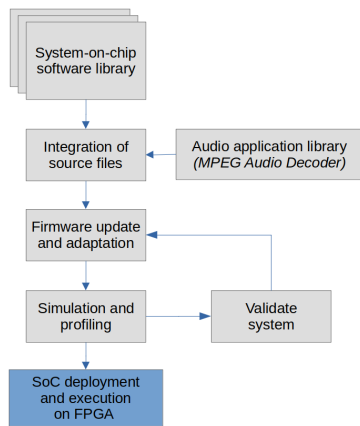


Figure 2: Software implementation flow diagram

The porting of the MAD decoder, originally composed in fixed-point, coded in ANSI C, entailed a task of code compilation, followed by running the program on a simulator. This step was instrumental in gathering crucial performance data, which was then analysed to identify potential areas for optimisation. Subsequently, this cycle of compilation, execution, and evaluation was methodically repeated in both development instances and involves ensuring the integration of these components within the broader SoC framework, which necessitates testing and validation procedures to confirm that the audio output meets the expected quality standards without compromising system performance. These iterative refinements aim to minimise resource utilisation while maximising the decoder's efficiency, making it well-suited for real-time audio processing tasks within the resource constraints of the SoC.

Deploying the specified application in the IOB-SoC is accomplished by integrating the source files that constitute the existing software segment with

the application's native library. One of the reasons for the successful deployment of application programs in the system-on-chip architecture is its adaptability: it is designed to be modular, allowing for seamless incorporation of software and hardware components. Furthermore, the architectural layout supports scalability, enabling the addition of more sophisticated functionalities or updates without significant rework, which is critical for maintaining technological relevance and accommodating future enhancements.

#### 3.2. LibMAD - MPEG Audio Decoder Library

The *LibMAD* [2], an acronym for "library of MPEG audio decoder", is an open-source project for high-quality decompression written in standard C. All three audio layers are fully implemented (layers I, II and III). It is distributed under the terms of GNU General Public License (GPL). It does not support MPEG-2 multi-channel audio, nor is it compatible with AAC. The proposed library provides the following characteristics:

- 24-bit PCM output
- 100% fixed-point (integer) computation
- Implementation based on the ISO/ IEC standards
- Hardware-agnostic, supports a collection of RISC ISAs
- Support for MPEG-1 (joint/ stereo channel mode) / 2 (LSF, single channel mode)

The results from MAD meet the computational accuracy standards required for compliance with ISO/IEC 11172-4. Typically, MAD functions as a complete Layer III ISO/IEC 11172-3 audio decoder, adhering to the standard's specifications. Optionally, MAD's configuration allows for adjustments in accuracy beyond the standard settings, balancing accuracy with performance.

For optimal performance, choosing an assembly language version of the fixed-point multiplication routines is advisable. Various assembly language versions have been developed for different CPU types.

MAD's subband synthesis routine, which is computationally demanding, could be made more efficient at the cost of slightly lower accuracy by modifying the fixed-point multiplication method using a small windowing constant.

Although this adjustment improves performance, and the accuracy reduction is typically imperceptible, it is not activated by default and must be manually enabled. Depending on the architecture, additional specific optimisations might also be available.

The repository version used was the *0.15.1b*, and the source code was modified only for code readability, debugging, and software behaviour profiling. By exploiting the PC emulation setting of the *IObundle, Lda.* system-on-chip, taking advantage of a virtual version regarding the physical components and peripheral drivers through adapted filesystem resources. The present case used a duration measurement for each function that composes the MPEG audio decoder software.

The decoder provides two modes of operation: synchronous and asynchronous. The synchronous mode operates on the stream sequentially using a single process, characterised by a blocking behaviour, making it impossible to perform other tasks coincidentally with decoding. On the contrary, the asynchronous use case is suitable when implementing non-blocking behaviour for concurrent processing systems and managing inter-process communication mechanisms. In this mode, different stages of the decoding process can be handled in separate threads or processes, allowing for parallel processing of other stream parts and improving performance, especially on systems with multiple cores or processors.

A synchronous approach is used in this case because the embedded systems environment does not have multi-thread support and relies exclusively on single-thread processing. In essence, the behaviour is reflected using a linear execution mode method that processes data consecutively, ensuring stable operation within these constraints.

In environments where resource constraints are significant, such as embedded systems with limited processing capabilities and memory, the predictability of a synchronous method greatly aids in minimising runtime errors and enhancing system reliability. It also reduces the overhead associated with complex thread management and synchronisation mechanisms, which are necessary in asynchronous systems.

Both MP1 and MP2 use similar encoding methods. They employ a method of audio compression that involves dividing the audio spectrum into small frequency bands and then encoding them. This similarity in the foundational practice makes it easier for an MP2 decoder to understand and decode MP1 files. Given the definition of audio layer N decoder, where N denotes layers I, II or III of the MPEG audio standard, it is backwards compatible with bit-stream data encoded in layer N and all layers below N. The analysis used MP2 and MP3 files to create a preliminary study of the decoder software's most intrinsic and demanding properties.

There are four main categories of execution concerning the decoding application:

- An initialisation phase, where the necessary

components and communication peripherals are defined and instantiated. The decoding routine's essential elements are the decoder, buffer and stream structures and functions for the high-level application programmable interface.

- The input stage is responsible for managing the flow of data acquisition and buffering into the decoder, as well as with the initial error detection and handling for the incoming data stream, ensuring the integrity of the data before processing.
- Decompression is the core of the decoding process, considering the audio synthesis and reconstructing the signal waveform from the decoded data into a raw audio format (PCM or WAV). It involves various sub-steps, such as Huffman decoding, inverse quantisation, and inverse discrete cosine transform (IDCT).
- The output stage involves sending the decoded audio data quantities to an output device and synchronising them with other transport channels or systems for correct playback timing. Handling the continuous flow of the audio stream to the output device ensures smooth and uninterrupted playback.

The MP3 decoding process involves the same functions for the synthesis and decompression of audio frames. The difference is that the decoding component is more prominent than the synthesis, which is notable in terms of complexity, where most of the computational load is found due to the increased resolution and fidelity in the compression process.

The process also applies Huffman and scale factor decoding, essential for deconstructing the compressed data into a form that can be converted back into audio signals. Huffman coding allows for efficient data compression without loss of information, while scale factors adjust the decoded audio data to match the original dynamic range. Moreover, MP3 decoding handles the potential for overlapping frames, frequency inversion, and alias reduction. Overlapping of frames can prevent audible gaps between frames, maintaining an ideal auditory experience. Frequency inversion and alias reduction are part of the process that ensures the output audio does not contain distortions that can arise from the encoding process.

The algorithmic functional routines are supported by the most basic mathematical/ logical operations, with tailored implementations regarding the instructions available for several central processing unit architectures, such as SPARC (Scalable Processor ARChitecture), PowerPC (Performance Optimisation With Enhanced RISC-Performance

Computing), MIPS (Microprocessor without Interlocked Pipelined Stages), ARM (Advanced RISC Machines), Intel and a 64-bit version (classified as the most accurate if the type is supported by the compiler, although not being the most efficient).

In this project’s scope, the default 32-bit fixed-point method is implemented, taking advantage of some RISC-V extensions and CPU architectures as a conveyor of improvement and adaptability of the audio decoding system requirements to real-time operation.

#### 4. Results

Evaluating an audio decoder’s performance involves several key metrics determining its suitability for real-time applications. These metrics reflect operational efficiency and highlight the system’s effectiveness in a real-world scenario.

Real-time audio decoding is crucial for applications such as streaming media, where delays between data receipt and audio output must be minimal. The decoder must process each audio frame within the frame’s duration to avoid playback interruptions or quality degradation, so the processing time requirements should be addressed.

A cumulative measure of the differences between the duration of each frame and the time taken to decode it, providing a measure of the system’s responsiveness. A positive value indicates that the decoder is operating within the required time limits, while a negative value would suggest a potential for real-time processing issues. A positive value indicates that the decoder operates within the required time limits, while a negative value suggests potential real-time processing issues. The elapsed execution time metric comparison between the CPU models used in the project to identify which delivers the fastest processing relative to the audio frame duration.

The efficiency of instruction sets and the number of operations dispatched per clock cycle are critical in minimising the decoding time. Many predominant factors, such as the processor’s operating frequency, affect how quickly instructions are processed, impacting the total running time overhead. Higher frequencies generally improve performance but can also increase power consumption and heat generation, which might not be ideal for all systems.

Ensuring the decoder meets real-time requirements involves the exploitation of instruction-level parallelism that varies according to the computing architecture and software optimisation. The basis for comparison relies on different processing unit options, according to elapsed decoding time and selecting the option that can handle the required operations efficiently within the time constraints of audio playback. This study evaluates the impact

of these factors on the decoding process by analyzing how the different architectures and RISC-V ISA extensions perform under the same workload at a constant frequency of 100 MHz.

#### 5. FPGA system resources utilization

Resource utilization of the MPEG decoder implemented on the FPGA is outlined in table 1. It compares it to the standalone MPEG decoder and the decoder running as part of a tester system. The focus is on several essential FPGA resources: Lookup Tables (LUTs), LUTRAM, flip-flops, Digital Signal Processors (DSPs), Block RAM (BRAM), Control Logic Block (CLB) and control signal sets.

Component	Resources	Utilization(%)	Total
LUT as logic	16568	6.83	242400
LUTRAMs	7711	6.84	112800
DSP	7	0.36	1920
RAMB36	90	15.00	600
RAMB18	1	0.08	1200
CLB	23042	4.75	484800
Control sets	1805	2.98	60600

Table 1: Resource utilization of different component types

#### 6. Profiling

The following table displays the execution times for various functions related to the MP2 decoding process, measured in microseconds ( $\mu s$ ), relative to the test sample group used for acknowledging different audio file types and features. The audio data sample group used for testing is divided by different audio file types defined in the subsection of the decoder verification:

Function	Noise	Speech	Jazz	Classical
<code>decode_header</code>	62	64	60	60
<code>mad_header_decode</code>	83	88	79	80
<code>mad_layer_II</code>	18537	18183	27012	25509
<code>II_samples</code>	30	31	31	30
<code>mad_bit_read</code>	3	3	3	3
<code>mad_synth_frame</code>	96914	96915	193733	193731
<code>dct32</code>	7878	7878	7878	7878

Table 2: Execution time per function related to the MP2 decoding procedure (frame average) [ $\mu s$ ]

The functions include data fetching, decoding algorithms, error checking, and output generation routines. Notably, the functions `mad_layer_II` and `mad_synth_frame` consume the most processing time during the MP2 decoding process. The `mad_layer_II` function handles decoding Layer II audio data, which involves complex mathematical operations such as subband synthesis and frequency inversion. The significant time spent for this function is due to the computational demand of processing and reconstructing the audio signal from its compressed form.

The table 3 displays the execution times for various functions related to the MP3 decoding process, measured in microseconds ( $\mu s$ ):

Function	Noise	Speech	Jazz	Classical
III_sideinfo	160	165	297	294
III_scalefactors	140	166	455	372
III_exponents	25	27	50	47
III_requantize	3	3	3	3
III_huffdecode	9342	9631	11413	10158
III_aliasreduce	5214	6514	6517	6517
III_overlap	712	712	980	1076
III_overlap_z	37	63	432	370
III_freqinver	144	151	288	298
fastsdct	50	51	63	63
sdctII	8029	8030	12218	13926
dctIV	11680	11680	17571	20256
imdct36	14547	12176	18527	21317
dct32	7878	7878	15720	15725
mad_synth_frame	89254	89254	178407	178407
mad_bit_read	1845	1858	1964	1964

Table 3: Execution time per function related to the MP3 decoding procedure (frame average) [ $\mu s$ ]

The group with the prefix "III" constitutes the bitstream decoding aspect; hence, the function's name represents the individual part of the procedure, which is notable for the significant execution times.

Jazz and classical recordings show increased execution times. This is attributed to the high bit-rate recordings, channel mode, complex harmonic content, and dynamic range, which require more intricate processing during the transform stages. Due to the complexity of the operations and the amount of processed data, most of the processing period is spent on functions related to mathematical transforms, decoding, and synthesis of audio frames.

## 7. Evaluation on different types of audio content

The system operates at a frequency of 100MHz and uses the PicoRV32 as the baseline for code profiling. Key aspects of the evaluation include the elapsed decoding time for comparison between the CPU models to identify which delivers the fastest processing relative to the frame duration.

Verifying the decoder's accuracy by comparing the expected versus actual decoded data ensures the implementation produces correct audio output. This is accomplished by validating the file contents, byte-by-byte, against the generated output made with the original implementation of LibMAD. All tested files yielded compliant results compared with the pure software implementation. Analysing decoder performance with different file specifications, such as bitrate, frequency, sample rate, and channel format, evaluates the algorithm's versatility in handling various audio content effectively.

To further assess the performance improvements attained with pipelined processing architectures, we compared the execution times of the PicoRV,

DarkRV, and VexRV RISC-V processing units across different types of audio content:

CPU	PICORV	VEXRV	DARKRV
Noise	115872	212306	5905 7047 3845 5778
Speech	139762	261403	8074 8241 5727 6465
Jazz	115177	183270	6056 6879 4043 4677
Classical	145453	272575	7083 8027 5315 6341

Table 4: Execution time on average per frame for MP2 and MP3 files [ $\mu s$ ]

PicoRV32 has significantly higher processing times than the VexRV and DarkRV, which is much more significant than the duration of one audio frame, making it impractical for reliable real-time use. In contrast, both the DarkRV and VexRV architectures achieve decoding times within the frame duration.

### 7.1. Real-time requirements

A positive halting difference value indicates that the each frame is decoded faster than the frame duration, leaving sufficient time for seamless audio playback. The DarkRV and VexRV CPUs consistently show positive values, demonstrating their ability to meet processing requirements. In contrast, the PicoRV32 shows negative values, confirming that it cannot process frames quickly enough for real-time decoding, in MP2 and MP3 file formats.

CPU	PICORV	VEXRV	DARKRV
Noise	-89750	-113640	+20217 +18048 +22277 +20395
Speech	-89055	-119331	+20066 +19039 +22079 +20807
Jazz	-186120	-235281	+19075 +17881 +20344 +19657
Classical	-157148	-246453	+19243 +18095 +21445 +19781

Table 5: Halting difference for decoding time per frame of MP2 and MP3 files [ $\mu s$ ]

The smallest discrimination thresholds for interaural time differences in human hearing are near 10  $\mu s$ . Therefore, the positive halting differences achieved by the DarkRV and VexRV CPUs (ranging from approximately +17,881  $\mu s$  to +22,277  $\mu s$ ) are well above this threshold, ensuring that the decoded audio will not introduce perceivable delays or synchronization issues. VexRV and DarkRV are compliant with the real-time processing requirement. The attained and required speedups for the audio decoding process values are summarized in the table below:

Audio file type	Required	Attained
Noise	4.44	5.35 30.14 24.40
Speech	1.60	4.04 29.96 27.37
Jazz	8.13	10.01 36.74 40.43
Classical	7.02	10.43 39.19 42.99

Table 6: Attained speedup and required values for MP2 and MP3 files

## 8. Conclusions

This chapter summarises the main achievements of the thesis, which focused on implementing an MPEG audio decoder on a System-on-Chip (SoC) with a RISC-V CPU. The project explored various facets of system composition, from software development and testing to the physical deployment of the system, emphasising flexibility and open-source code integration.

### 8.1. Achievements

The decoder's implementation demonstrated high flexibility in handling different audio formats with compliance by exploiting processing parallelism and significantly improving decoding efficiency. The approach to achieving the desired performance requisites was taken to the level of the central processing unit.

Digital audio data decompression is successfully achieved with the PicoRV architecture and efficient real-time decoding with the DarkRV and VexRV units. It also met stringent sound quality compared to pure software implementation, with reduced component usage in the FPGA fabric. The project highlighted the challenges and solutions in moving from a simulated environment to actual physical implementation. Several key deliverables were produced:

- The open-source *LibMAD* library was ported to the MPEG decoder system, and comprehensive execution profiling was conducted to analyze CPU cycle count during the decoding procedure to validate the real-time processing requirement.
- I/O modules to transport encoded and decoded audio data, fundamental for incorporating the decoder into existing infrastructures, providing seamless data flow for real-time audio decoding.
- Connection with a tester platform, allowing comprehensive testing and validation under simulated real-world conditions. This setup was essential for verifying the performance and compliance of the decoder.
- Achieved enhancements at an operational frequency of 100 MHz without compromising audio decoding accuracy, compared with the pure software implementation of the decoder.

### 8.2. Future work

While the project met its primary objectives, the following areas are suggested for future development:

- Expanding the decoder to support additional audio formats would increase its applicability, for example, AAC or FLAC.

- Integration with an operating system to exploit asynchronous digital audio decoding.
- Burst-mode option and multiple channels support.
- Non-isolated implementation on a fully working environment with an encoder source and decoder endpoint on a DAB stream.

This study contributes to the ongoing efforts to develop more efficient, scalable, and adaptable solutions for audio decoding in MPEG-1/2 formats, which can conceive and enhance audiovisual applications across broadcasting and multimedia, extending to future digital communication frameworks.

## References

- [1] P. L. Yingbiao Yao, Qingdong Yao and Z. Xiao, "Embedded software optimization for MP3 decoder implemented on RISC core," *IEEE Transactions on Consumer Electronics*, vol. 50, 2004.
- [2] I. Underbit Technologies, "LibMAD - MPEG audio decoder library," 2004. Accessed December 2023.
- [3] R. G. David Katz, *Embedded Media Processing*, ch. 5, p. 65. Elsevier Science & Technology, 2005.
- [4] S. W. Smith, *Guide to digital signal processing*, ch. 8, 9, pp. 141–177. California Technical Publishing, 1999.
- [5] A. Sugiyama and M. Iwadare, "The origin of digital information devices: the silicon audio and its family," *APSIPA Transactions on Signal and Information Processing*, vol. 7, 2018.
- [6] W. Su and P. Razaghi, "Design of flexible audio processing platforms using the system-on-chip environment," tech. rep., Department of Electrical & Computer Engineering, The University of Texas at Austin, 2012.
- [7] S. M. Resve Saleh, Steve Wilton, "System-on-Chip: Reuse and Integration," *Proceedings of the IEEE*, vol. 94, pp. 1050–1069, June 2006.
- [8] IObundle, "IOb-SoC," 2020. Accessed May 2024.
- [9] F. H. Dirk Koch and D. Ziener, *FPGA for Software Programmers*, ch. 15, pp. 261–284. Springer International, 2016.
- [10] K. A. Andrew Waterman, "The RISC-V instruction set manual - volume I: User level ISA," tech. rep., CS Division, EECS Department, University of California, Berkeley, 2017.