MPEG1/2 layers I/II encoder using a RISC-V processor and hardware accelerators

Tiago Alves da Silva
Instituto Superior Técnico
Universidade de Lisboa
Lisboa, Portugal

***Abstract*—This work introduces an MPEG-1/2 Layer II Audio encoder for a RISC-V embedded architecture featuring a reconfigurable hardware accelerator. Although such systems are standard on commercial embedded processors such as ARM, this work is the first to present a RISC-V implementation. The advantage is that the RISC-V architecture is an open specification with a few open-source hardware designs available. A hardware accelerator allows the system to run on low-frequency environments like an FPGA device. In this work, the system software uses the TwoLAME encoder open-source library. The system hardware is based on IOb-SoC, an open-source RISC-V SoC platform written in Verilog. The VexRiscv CPU has been chosen, and the hardware accelerator has been implemented using the Versat open-source reconfigurable accelerator design tool. The work features software optimizations and two hardware accelerators to accelerate the computation of the psychoacoustic model of the algorithm. The base performance is 6.2x slower than real-time for a system running at 100MHz, which indicates that an implementation for 620MHz would meet the goal. With hardware acceleration, the achieved performance is 2.4x slower than real-time for a system running at 100MHz, which indicates that an implementation for 240MHz would meet the goal.**

***Index Terms*—Audio encoder, *TwoLAME*, System-on-Chip, Hardware acceleration, Field-Programmable Gate Array.**

I. Introduction

A. About MP2

MPEG-1 Layer II, often referred to as MP2, is an audio compression format developed as part of the MPEG-1 standard. MPEG-1 is a set of standards created by the Moving Picture Experts Group (MPEG) for video and audio compression.

MP2 is less well-known and widely used than its successor, MP3, which offers higher compression and better audio quality for consumer applications. However, MP2 has been used in various professional applications, including: **Digital Broadcasting**, **Digital Audio Recording**, **Video DVDs**, **Video Conferencing** and **Archival Audio**.

B. Motivation

There are three options for integrating an MPEG-1/2 Layer II encoder in a system: buy an encoder chip, use a software encoder in the user system, or buy an IP core.

Encoder chips, like the *CX23415 MPEG-2 Codec* [1], the *MPEG-2 Encoder CW-4888* [2], or the *Futura II ASI+IP* [3] can be purchased off the shelf. However, this means an additional chip on the board, increasing its area/volume, weight, and power consumption.

Software encoders are commonplace in commercial embedded processors, but it is essential to consider the cost and effort associated with porting the software to the CPU.

The last option is to license a commercial MP2 encoder IP Core. It reduces area/volume, weight, and power consumption, allowing users to develop a top-notch system. To the best of the author's knowledge, the only IP core in the market is the *CWda74* [4], later re-branded *IPB-MPEG-SE* [5], which uses fixed-point calculations.

Therefore, this work considers the design of a RISC-V IP core. Some reasons for such a design are:

- Open-Source and Customizable Architecture
- Low Power Consumption
- Integration and System-on-Chip
- Custom Instructions and Accelerators
- Licensing and Cost

C. Objectives

This work introduces an MPEG-1/2 Layer II Audio encoder for a RISC-V embedded architecture featuring a reconfigurable hardware accelerator. The encoder will be developed using *IObundle, Lda*'s [6] IOb-SoC [7], a System-on-Chip template comprising an open-source RISC-V [8] processor. The TwoLAME [9] repository, an open-source MP2 encoding software based on the ISO/IEC 11172 [10], will provide the algorithm. The objectives of this work are the following:

- Port the *TwoLAME* [11] encoder open-source library to the system, using floating-point precision.
- Perform software optimizations.
- Profile the *TwoLAME* algorithm while running in FPGA [12].
- Use *Versat* [13] open-source reconfigurable accelerator design tool to accelerate *TwoLAME* algorithm.
- Check the requirements for *TwoLAME* real-time encoding.
- Compare with the fixed-point *CWda74* [4], the only competitor.

II. Background

A. ISO/IEC 11172-3: 1993 (E)

The ISO/IEC 11172 is an international standard under the title *Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s*, which specifies the coded representation of high-quality audio for storage media, among other things.

1) Audio encoder: An audio encoder processes digital audio to create a compressed bitstream for storage, meaning that the encoder's output should enable a compatible decoder to generate suitable audio for the intended purpose. Figure 1 illustrates the basic structure of an audio encoder [10].

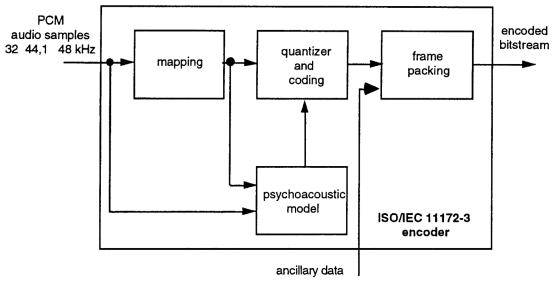

Fig. 1: Audio encoder basic structure.

First, the **mapping** block creates a filtered and subsampled representation of the input audio stream called subband samples. At the same stage, the **psychoacoustic model** block creates a set of data to control the next block.

Then, the **quantizer and coding** block creates a set of coding symbols from the mapped input samples.

Finally, the **frame packing** block assembles the actual bitstream from the output data of the previous blocks.

The encoding process supports single-channel, dual-channel, stereo, and Joint Stereo modes.

2) Psychoacoustic encoder: Figure 2 illustrates the primary parts of *psychoacoustic* algorithm.

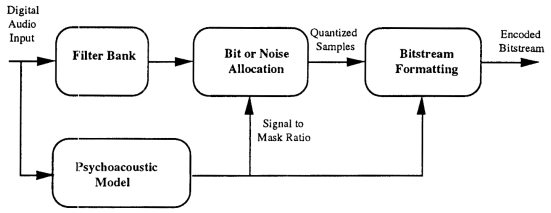

Fig. 2: ISO/IEC 11172-3 encoder block diagram.

The **Filter Bank** does a time-to-frequency mapping, being either a *polyphase* or a hybrid *polyphase/Modified discrete cosine transform* (MDCT) [14] filter bank. These filterbanks are critically sampled, having the same number of samples in both analyzed and time domains, and provide the primary frequency separation for the encoder, with quantized output samples.

The **Psychoacoustic Model** calculates a just noticeable noise level for each band in the filter bank. This noise level is used in the *Bit or Noise Allocation* part to determine the actual quantizers and quantizer levels. The final output of the model is a signal-to-mask ratio (SMR) for each band.

The **Bit or Noise Allocation** takes both the output samples from the *Filter Bank* and the SMR from the *Psychoacoustic Model* and adjusts the bit allocation, to meet the bitrate and masking requirements.

The **Bitstream Formatting** takes the *quantized filterbank* outputs, together with the bit allocation and other required side information, and encodes and efficiently formats all that information.

B. MPEG-1/2 Layers I/II Software

1) TwoLAME: TwoLAME is an optimized MP2 audio encoder, based on *TooLAME*, with its latest version released in 2019. Table I describes some features provided by *TwoLAME*.

Features
Static and shared library (*libtwolame*)
Fully thread-safe
API similar to *LAME*'s (easy porting)
Front-end supports a wider range of input files
Written in Standard C (ISO C99 compliant)

TABLE I: *TwoLAME* features.

The *TwoLAME* repository includes a *simplefrontend* directory that contains a basic implementation of the algorithm, written in C. Figure 3 shows the pseudo code for the first part of *simplefrontend.c*, mainly consisting of initialization.

```
Beginning of main function

    Allocate memory for the PCM audio data
    Allocate memory for the encoded MP2 audio data
    Initialize TwoLAME encoder and set default parameters
    Open the WAV input file
    Set the number of channels in the input stream
    Set the MPEG audio mode for the output stream
    Set the sample rate of the PCM audio input
    Set the sample rate of the MPEG audio output
    Set the bitrate of the MPEG audio output stream
    Validate parameters and allocate internal buffers
    Open the MP2 output file
```

Fig. 3: First part of *firmware.c* pseudo code.

The program starts by allocating memory for two different buffers. One buffer is *pcmaudio*, which receives part of the original input file. The other buffer is *mp2buffer*, which receives part of the encoded file.

The first function is *twolame_init*, which initializes the encoding software by setting defaults for all parameters. The second function is *wave_init*, which parses the wave header obtaining information like sample rate and audio mode. The remaining initialization functions specify encoding options, namely *twolame_set_num_channels*, *twolame_set_mode*, *twolame_set_in_samplerate*, *twolame_set_out_samplerate* and *twolame_set_bitrate*.

After this, *twolame_init_params* function validates all the parameters and initializing internally used variables. Then, *fopen* opens the output file where the encoded MP2 data is later written.

Figure 4 shows the pseudo code for the second part of *simplefrontend.c*, mainly consisting of the encoding process.

This process is based on a *while* loop, as the main idea is to encode one part of the input audio file at a time. Each loop iteration starts by reading a chunk of data from

```
Beginning of loop

    Read audio data samples from the WAV input file
    If there is any sample to read:
        Encode PCM audio data to MP2 using TwoLAME
        Write the MPEG bitstream to the MP2 output file
    If not:
        Exit from loop

End of loop
```

Fig. 4: Second part of *firmware.c* pseudo code.

the input file to the PCM buffer. With the buffer already loaded, *twolame_encode_buffer_interleaved* is responsible for encoding the audio data using *libtwolame*. After encoded, the data is written in the output file.

Lastly, figure 5 shows the pseudo code for the third part of *simplefrontend.c*.

```
Encode any remaining PCM audio data to MP2
Write the MPEG bitstream to the MP2 output file
Shut down the TwoLAME encoder and free all allocated memory

End of main function
```

Fig. 5: Third part of *firmware.c* pseudo code.

In this part, the algorithm finishes execution. The first function is *twolame_encode_flush*, which encodes any remaining buffered PCM audio, returning at most a single frame. The remaining encoded data is written in the output file. Lastly, all memory is freed.

C. System-on-Chip

A System-on-Chip (SoC) is an integrated circuit that combines components of an electronic system. The IOb-SoC is a System-on-Chip template, comprising an open-source RISC-V [8] processor, which users can modify, simulate, and implement in ASICs [15] and FPGAs [12]. Figure 6 shows a base IOb-SoC high-level block diagram [16].

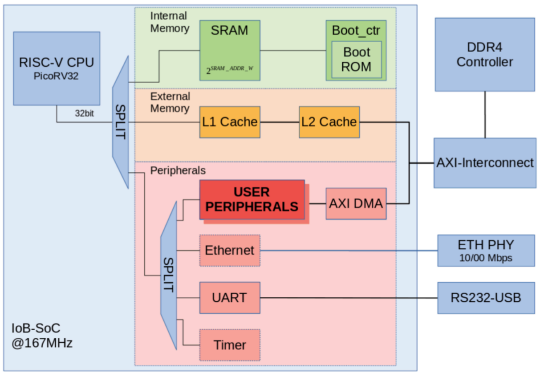

Fig. 6: Base IOb-SoC high-level block diagram.

1) IOb-SoC components: The IOb-SoC uses a PicoRV32 CPU core, an open-source 32-bit processor that implements the RV32IMC instruction set.

As **Memory** subsystem, this SoC includes three main components. The Boot Read-Only Memory **Boot ROM** is a ROM used for booting the system. The Static Random Access Memory **SRAM** [17] is an internal memory that allows the system to run the program or the bootloader. The **Cache** is an optional component that stores data from the external Double Data Rate (DDR) memory.

The **Memory bus** allows the CPU to communicate with the memory subsystem, while the **Peripheral bus** allows the CPU to communicate with peripherals. The **IOb-Interconnect** is a bus switch responsible for the valid-ready handshake protocol between the CPU and all the peripherals.

Finally, The Universal Asynchronous Receiver-Transmitter (**UART**) peripheral allows the SoC to communicate with external systems, through the RS-232 serial communication protocol. All these components are integrated into IOb-SoC as GitHub submodules.

D. Versat

FU	Description
Const	Outputs a 32-bit value configurable by the CPU.
FloatAdd	Adds two 32-bit floating-point inputs.
FloatSub	Subtracts two 32-bit floating-point inputs.
FloatMul	Multiplies two 32-bit floating-point inputs.
FloatNot	Negates the most significant bit (MSB) of the 32-bit floating-point input.
Float2Int	Converts a 32-bit floating-point input into a 32-bit integer value.
Mux2	Operates as a 2-to-1 multiplexer, selecting one of the 32-bit inputs based on the 1-bit control input.
Mem	Contains an internal memory with two input and two output ports. It offers a memory-mapped interface that allows the CPU to store and read data from the memory while the *Versat* accelerator is not running.
LookupTable	Contains an internal memory with two input and two output ports (Dual port synchronous RAM). It offers a memory-mapped interface that allows the CPU to store and read data from the memory while the *Versat* accelerator is not running.

TABLE II: *Versat* functional units.

Versat is a Coarse-Grained Accelerator designed for embedded systems. It addresses the challenge of accelerating compute-intensive inner loops in code while efficiently managing the transitions between code suitable for the Coarse-Grained Accelerator and code that needs to run on the host processor [18]. A list of the key features is presented below:

- **Partial Reconfiguration**: *Versat* uses a configuration register file and a configuration memory.
- **Integration with Embedded Systems**: *Versat* cores serve as co-processors within embedded systems, working alongside application processors.
- **Compiler**: *Versat* acts like a compiler. The programming language syntax is a subset of C/C++, with hardware data descriptions.

1) Functional units: Versat contains several functional units (FUs) in its source, all written in Verilog. A brief description of each functional unit is provided in the table IV.

2) Operators: Versat also contains a set of operators. This terminology just abstracts the implementation through functional units. A brief description of some operators is provided in the table below.

Operator	Description
$-$	Negates the right operand.
$\sim$	Performns binary one's complement of the right operand.
$\&$	Performs a logical AND between the left and right operands.
$\ll$	Shifts the bits of the left operand to the left by the number of positions defined by the right operand.
$\vert$	Performs a logical OR between the left and right operands.
$\wedge$	Performs a logical XOR between the left and right operands.

TABLE III: *Versat* operators.

3) Syntax: Figure 7 shows a coding example of *versatSpec.txt*, the file where the *Versat* accelerator should be described.

```
module Example (op1, op2) {
    FloatMul mul;
#
    op1 -> mul:0;
    op2 -> mul:1;
    mul -> out;
}

module top () {
    Const A;
    Const B;
    Mem mem;
    Example ex;
#
    add = A + B;
    add -> mem:0;

    sub = A – B;
    sub -> mem:1;

    mem:0 -> ex:0;
    mem:1 -> ex:1;
    ex -> out;
}
```

Fig. 7: *versatspec.txt* example.

The code is divided into two modules, *Example* and *top*.

The *Example* module is defined with two input ports, *op1* and *op2*, and a *FloatMul* FU, *mul*. The code after the # symbol specifies how data flows through the module. In this case, *op1* and *op2* are connected to input ports 0 and 1 of the *mul* FU, respectively. Then, the output of the *mul* FU is connected to the output port of the *Example* module.

The *top* module is the starting module, i.e. it represents the overall design. This module is defined with two *Const* FUs, *A* and *B*, and a *Mem* FU, *mem*. In addition, it also contains an instance of the *Example* module called *ex*. Once again, the code specifies how data flows through the modules. In this case, *A* and *B* are added together, and the result is stored in the *add* signal. Then, *add* is connected to input port 0 of the *mem* FU. *A* and *B* are also subtracted, and the result is stored in the *sub* signal. Then, *sub* is connected to input port 1 of the *mem* FU. Afterward, the output ports 0 and 1 of *mem* are connected to the input ports 0 and 1 of *ex* instance, respectively. The *ex* instance performs what was described previously for the *Example* module, and its output is connected to the output port of the top module.

III. Hardware architecture

In the initial stage, the IOb-MP2-E system comprised six main components: **AXI**, an AXI interconnect protocol; **CACHE**, a high-performance Verilog cache; **MEM**, a set of memory Verilog descriptions; **PICORV32**, a RISC-V processor; **UART**, a UART core; **DDR4 Controller**, a controller for DDR memory (figure 8). The ultimate goal of this work is to develop a system that beats the *CWda74* [4]. This system uses fixed-point precision, and so the most logical approach to the problem is implementing the *TwoLAME* algorithm using floating-point.

A. VexRiscv

Due to its fixed-point arithmetic limitation, the **PICORV32** CPU had to be substituted, and the **VEXRISCV** became the premier pick, as it includes a floating-point unit (FPU) [19]. Apart from removing **PICORV32** and adding the **VEXRISCV** to IOb-MP2-E, some interrupt signals were added to the system data bus.

B. Versat

The next step was adding **VERSAT** to the IOb-MP2-E (figure 8). This required adding an AXI interconnection between *Versat* and external memory. The system usually contains a single AXI [20] interconnect instance, used by external memory and CPU. Therefore, a pragmatic solution was to double the wire size of the existing AXI interconnection, providing communication from/to *Versat* while keeping the same AXI instance.

1) Functionality and Interfaces: Focusing on the *Versat* matter, *versatSpec.txt* functions like a code script, while *Versat* operates akin to a compiler, generating the accelerator in Verilog and the necessary headers and C source code for a correct correspondence between CPU and accelerator.

Versat generates hardware accelerators according to the dataflow paradigm. It instantiates functional units and connects them to execute the intended part of the algorithm. *Versat* handles data validity transparently from the user, inserting buffer units between connections to ensure data from convergent paths arrive at the same time when needed. The FUs need to implement certain interfaces recognizable by *Versat* to perform useful work, like input and output ports, as well as interfaces that allow them to send/receive data from/to outside the accelerator:

- **Configuration:** Inputs that allow the CPU to modify and control the functionality of individual units. A configuration register connects to the configuration entries of each unit, conditioning its behavior.
- **State:** Outputs that allow the CPU to read data from the units. Useful for small amounts of data transfer.

- **Memory-Mapped:** Allocation of specific address space for units, allowing access to them through read and write operations. This interface is primarily used for memory units, enabling the CPU to read and write data as required.

The accelerator generated by *Versat* groups all these interfaces into a single memory-mapped interface, accessed by the CPU. When the CPU accesses an address inside the accelerator memory space, the CPU can access a control register of the accelerator, the configuration, and the state registers or it can access the FUs directly through the memory-mapped interface. A DMA mechanism (configured by the CPU) is used for efficient data transfers between the accelerator and external memory.

C. Timer

To allow profiling the *TwoLAME* algorithm, Timer was added to IOb-MP2-E. This peripheral is a 64-bit hardware timer, equipped with reset, enable, and reading functions. Figure 8 shows a high-level block diagram of the IOb-MP2-E system at this point.

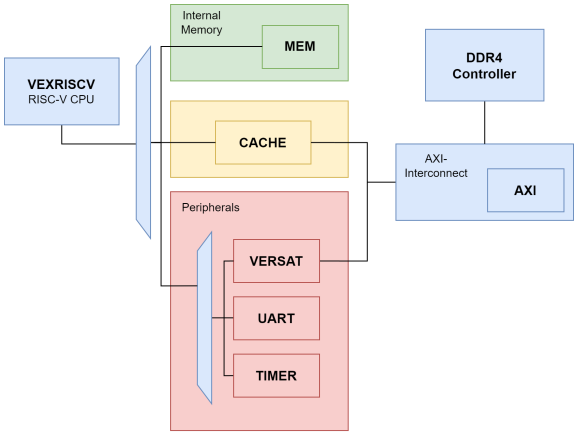

Fig. 8: High-level block diagram of the IOb-MP2-E in use.

Based on the results of *TwoLAME*'s profiling, shown in *Results* section, the subsequent step involves developing the hardware accelerators for the *psycho_3_threshold* function. The *psycho_3_threshold* function contains three for loops, with the first one being a simple initialization process, setting each element of arrays *LTtm* and *LTnm* to *DBMIN*. Performing this in hardware incurs unnecessary overhead since DBMIN can be provided as input for the subsequent hardware operations. Therefore, only two accelerators should be developed to execute the second and third for loops of *psycho_3_threshold*. The third for loop is simple as it just includes the *psycho_3_add_db* function. The second for loop is not only complex but it is also more interesting. The outer loop contains two *if* conditions and, inside each condition, there is an inner loop differing only on part of the input data. This means that it is only necessary to develop hardware that executes both inner loops, with the if conditions and the outer loop being handled by the CPU.

Before developing the accelerators, it is crucial to determine the control and data paths of the loops that are intended to be accelerated. The control path defines the flow and sequencing of operations within the software that need to be accelerated. The data path refers to the route through which data flows within the software during its execution.

D. spectrum_search accelerator

1) Control and Data paths: This section shows both control and data paths of the first accelerator, *spectrum_search*, corresponding to the second for loop in the original *psycho_3_threshold* function.

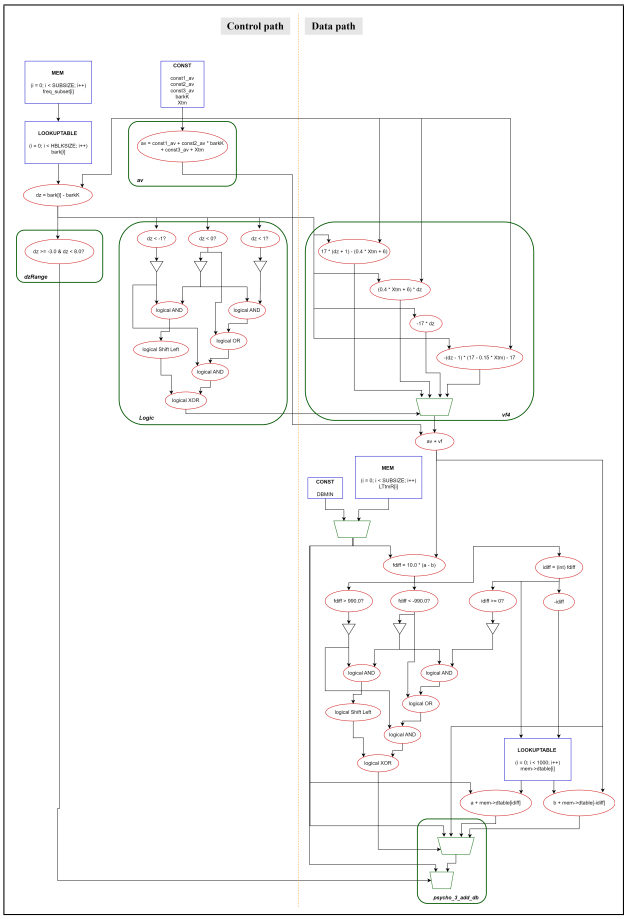

Fig. 9: Control and data paths of the *spectrum_search* hardware accelerator.

2) Functional units: Based on the previous Control and Data path, five new FUs were designed to allow the developing of the *spectrum_search* accelerator, as shown in figure IV.

3) Modules: After analyzing the control and data paths, the hardware accelerator is developed in *versatSpec.txt*. This file specifies the whole data process, which can be divided into several modules each representing a certain functionality or output. The starting module is called *start*, which interconnects all modules (and FUs) by setting the correct inputs and outputs, namely ***av***, ***dzRange***, ***Logic***, ***vf4*** and ***psycho_3_add_db***.

4) Control: Setting all the input data for the hardware accelerator is also required.

The first accelerator executes both inner loops in the second for loop of *psycho_3_threshold*. Since it does several runs,

FU	Description
FloatLess	Performs a 32-bit floating-point comparison operation to determine if the first input is less than the second.
FloatGreater	Performs a 32-bit floating-point comparison operation to determine if the first input is greater than the second.
FloatGreaterEqual	Performs a 32-bit floating-point comparison operation to determine if the first input is greater than or equal to the second.
Mux4	Operates as a 4-to-1 multiplexer, selecting one of the four 32-bit inputs based on the two least significant bits (LSB) of the 32-bit control input.
Conditional1	Operates as a 2-to-1 multiplexer, selecting one of the 32-bit inputs based on the 32-bit control input.

TABLE IV: New *Versat* functional units.

there is the need to supply the accelerator with different sets of input data. This is done in two separate functions because part of the data is valid for all the runs.

In *initVersat* the constants are set by assigning the intended value to the constant name (in the source files generated by *Versat*). For floating-point data, the *PackInt* function is required. There is also a memory unit setting for *freq_subset* array. This sets an internal memory with each position of the array, from 0 to SUBSIZE, read by the CPU through *VersatUnitWrite* function, sequentially. There are also two memory unit settings for *bark* and *mem→dbtable* arrays. These set two lookup tables, with SUBSIZE and HBLKSIZE, respectively.

In *configureVersat* four constants are set as they differ between runs. An input of *mux2* multiplexer is also set based on a flag that is handled by the CPU. At this point, all the input data is set for the hardware accelerator, and so it can start the execution through *RunAccelerator* functions. This function receives as an argument the number of times that the accelerator should run with the same settings, which is 1 in this case.

E. masking_threshold accelerator

1) Control and Data paths: This section shows both control and data paths of the second accelerator, *masking_threshold*, corresponding to the third for loop in the original *psycho_3_threshold* function.

2) Modules: The starting module is called *star1*, which interconnects ***psycho_3_add_db_V1*** module (and all FUs) with the main module.

start1

This module has some similarities with the *start* module of the first accelerator since it contains most of the *start* module description but is duplicated. It invokes and interconnects all the other modules, apart from executing simple logical and conditional operations.

3) Control: The second accelerator executes the third for loop of *psycho_3_threshold*. The control of this accelerator is done inside a single function since it performs only one run. Therefore, all the settings belong to *configureVersat1* function.

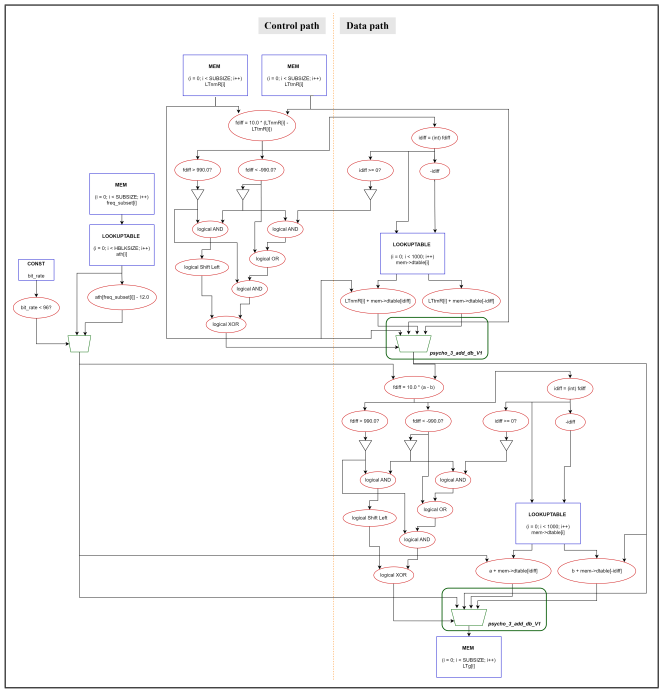

Fig. 10: Control and data paths of the *masking_threshold* hardware accelerator.

In this function, the process is similar to the previous accelerator, since the *masking_threshold* accelerator does not contain any functional unit that is not present in the first one.

F. Overview

The process of creating IOb-MP2-E involves synthesizing and implementing the entire system configuration within the FPGA, i.e. it defines how the hardware components are mapped and interconnected within the FPGA fabric.

Once the system configuration is established, the firmware becomes a vital component. The firmware is converted into machine instructions specific to the CPU RISC-V architecture. After synthesis and compilation, the firmware is stored in a Block RAM in the FPGA. When the FPGA is powered on or reset, the firmware is loaded from this memory and executed by the CPU.

An overview of the hardware setup employed in this work is presented below.

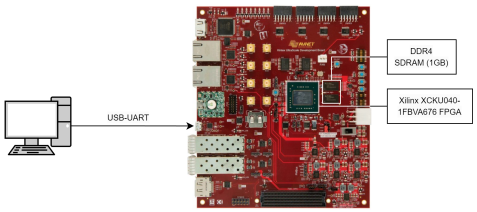

Fig. 11: Hardware setup overview.

IV. Software Architecture

Before being ported to the IOb-MP2-E, the *TwoLAME* software was tested in a Linux [21] environment, allowing

one to verify its functionality independently before integration. This verification consisted of listening to both an original audio file and the corresponding encoded file, produced by *TwoLAME*, since the *TwoLAME* software had already been tested and approved by a wide range of users.

A. Audio test files

In this context, *Audacity* software [22] was used to generate four audio files. The creation of multiple testing files was less about functionality verification and more about preparing for the algorithm's testing in the upcoming phases of this work, motivated by the following ideas:

- **Variation in Audio Characteristics**: Testing files with diverse audio characteristics allow a comprehensive assessment of *TwoLAME*'s adaptability.
- **Consistency Across Specifications**: Maintaining consistent software performance across various specifications is crucial for a real-time encoding IP core.

The first generated file was *short.wav*, using the *Generate Tone* option in *Audacity*. This mono audio has a sine waveform and a size of 27KB (the smallest audio file in the repository). The second generated file was *long.wav*, using the *Generate Rhythm Track* option configured with *Metronome Tick* beat sound in *Audacity*. This mono audio has a size of 683KB. The third generated file was *noise.wav*, using the *Generate Noise* option configured with *White* noise type in *Audacity*. This audio file has a size of 529KB, being stereo. The mono default track was duplicated, with one being distributed 100% to the Left and the other being distributed 100% to the Right (*panning effect*). The fourth and last generated file was *vivaldi.wav*. This file was not generated from scratch in *Audacity*. Instead, it was simply converted to WAV format. The original file, entitled 'Vivaldi - Spring', was downloaded from the Internet (for free) and opened on *Audacity*. After that, the track was reduced to the initial 4 seconds and exported as WAV, being also stereo. All the files have a 44.1KHz sampling rate.

At this point, it was already possible to verify the *TwoLAME* software. The next step was porting *libtwolame* (*TwoLAME*'s library) to the IOb-MP2-E, which required a front-end interface.

B. Firmware

Based on the *simplefrontend* example present in *TwoLAME*'s repository, the front end was implemented in *firmware.c*. As the name indicates, this file specifies the firmware that runs on the CPU as a software application which, in this work, is intended to be the *TwoLAME* algorithm. Figure 12 shows the pseudo code for the *main()* function in *firmware.c*.

The first difference in *main()* is the usage of *uart_init*, which resets *IObundle, Lda*'s UART peripheral. There are also two additional memory allocations for the input and output data. After allocating memory, the input audio data is received via UART in *uart_recvfile*. The IOb-MP2-E does not have an operating system or file system, and so the

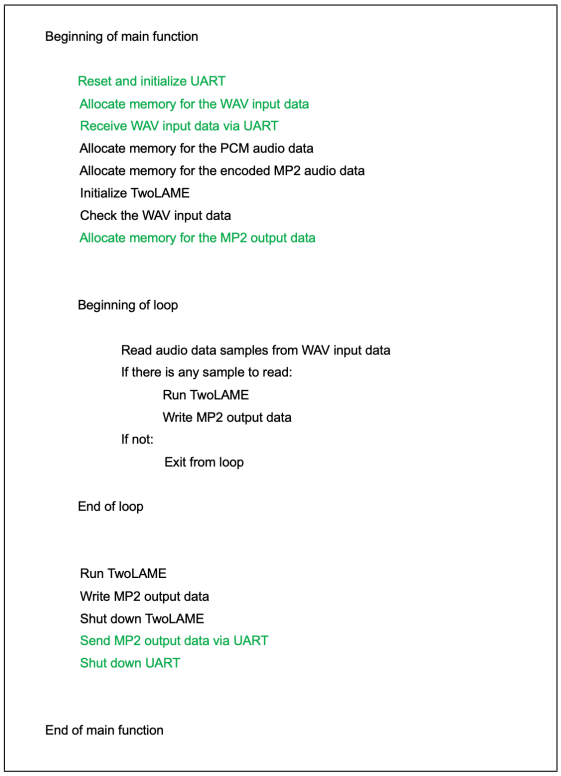

Fig. 12: New *firmware.c* pseudo code.

fopen function was removed. Furthermore, in each iteration of the *while* loop, a chunk of input data is read from the input data using *fread*. Since this function also requires a file system, there should be some process to increment the pointer to the input data, so that *fread* can be removed. This is achieved by decrementing a global variable based on the number of samples read in each loop iteration. In addition, since the *pcmaudio* buffer should contain short integers, there is a process inside *wave_get_samples* that converts every two characters into a single short integer. This is done by concatenating two characters, with the first one shifted left eight positions.

Nonetheless, not only the input audio data is received but also the output audio data is sent to IOb-MP2-E's external memory, both via UART. Thus, the *fwrite* functions were also removed. Another modification in *main* consisted of sending the output data via UART in *uart_sendfile* and closing the UART transmission through *uart_finish*.

Apart from *main*, some header files were included in *firmware.c*. Two macros were also defined, AUDIOBUFSIZE as 2304 and MP2BUFSIZE as 4096. These variables specify the size of input and output buffers, respectively, which in practice represents the chunk of data encoded each time (the number of frames encoded). Therefore, the number of frames can be altered by changing both variables on the same scale. As it stands, the *TwoLAME* encodes one frame at a time. After this process, the system was emulated on PC, passing through a simulation environment developed by *IObundle, Lda*, which allowed full testing.

C. Optimization

Some basic software optimizations were performed. The first change in *libtwolame* was the software precision. In *comon.h*, there is a macro for FLOAT that was initially defined as *double*, but it was changed to *float*. One function in *subband.c* was also changed. The original code contained *modf* [23], a function that belongs to *math.h* [24] and requires double precision. Therefore, it was substituted by *modff* [23], which has the same functionality but uses single precision.

The second change was more strategic as it involved substituting several trigonometric functions by recomputed lookup tables, for specific input ranges. These cases occured in *psycho_3_powerdensityspectrum*, *psycho_3_init_add_db*, *psycho_3_spl*, *psycho_3_fft* and *create_dct_matrix* functions.

The third and last change was related to memory, as the *bit_stream* struct memory was being allocated in every *while* loop iteration, in *main*. After this process, the system was emulated on the PC once again to check correctness.

The next move was measuring the execution time of IOb-MP2-E in FPGA, for different input data. This requires different memory management since allocating memory in an FPGA typically needs to be done statically, rather than dynamically during program execution.

D. Profiling

The last implementation in the software architecture was **profiling** [25], used to measure the duration of each *TwoLAME* function. The process of profiling *TwoLAME* consisted of three phases. In the first phase, *twolame_encode_buffer_interleaved* was analyzed. Then, the second phase of profiling included all function calls inside *encode_frame*. In the third and last phase of profiling, *twolame_psycho_3* was analyzed (results in table V).

V. Results

A. Profiling

Table V shows that*psycho_3_threshold* occupies the biggest part of the program execution. For *short.wav*, *long.wav*, *noise.wav* and *vivaldi.wav*, this function corresponds to 37%, 61%, 64% and 63% of *TwoLAME* execution time, respectively.

B. FPGA implementation

All the work was tested using *KU040 Xilinx Kintex® UltraScale™ Development Kit* [26]. In this work, the Xilinx board runs at 100MHz, the frequency at which IOb-MP2-E is compiled. Table VI shows the implementation results of IOb-MP2-E.

Focusing on the measurement without *Versat* and with *Versat* using *spectrum_search* accelerator, the results show a significant increase in Total LUTs, Flip-Flops and DSP Blocks.

C. Execution time

Table VII shows that the *spectrum_search* accelerator can induce a substantial reduction in execution time. Specifically, the *psycho_3_threshold* execution times for *short.wav*, *long.wav*, *noise.wav* and *vivaldi.wav* are reduced by approximately 96.26%, 94.6%, 95.34% and 94.98%, respectively. With this reduction, the *masking_threshold* accelerator becomes unnecessary. That is because *Versat* does not allow the execution of different accelerators in the same run. The *masking_threshold* accelerator did not achieve any acceleration at all

D. Real-time requirements

After measuring the execution time of the original and the hardware-accelerated implementation in IOb-MP2-E, it is possible to calculate the speedup achieved for each input file, using the formula presented below.

$$\textbf{\textit{Speedup achieved}} = \frac{\textit{Execution time without Versat}}{\textit{Execution time with Versat}} \quad (1)$$

Additionally, the *Amdahl's Law* [27] can be applied to understand the limitations of the hardware acceleration, for the *psycho_3_threshold* function. The ideal case where the speedup (s) is very high was considered.

$$\textbf{\textit{Speedup desired}} \approx \frac{1}{1-p} \quad (2)$$

A more interesting point is to compare the achieved speedup with the desired speedup based on the real-time requirements, for each input file.

$$\textbf{\textit{Real-time}} = \frac{\textit{number of frames} \times \textit{number of samples}}{\textit{sampling frequency}} \quad (3)$$

In this case, the real-time is calculated from the number of frames, the number of samples, and the sampling frequency.

The *sampling frequency* is $44100Hz$ and the *number of samples* is 1152, for all input files. The files *short.wav*, *long.wav*, *noise.wav* and *vivaldi.wav* contain 11, 296, 114 and 162 frames, respectively. Table VIII shows the real-time for all input files.

	Input file			
	short.wav	*long.wav*	*noise.wav*	*vivaldi.wav*
Real time	287	7732	2978	4232

TABLE VIII: Real time for all input files encoding [ms].

Based on the previous information, the speedup required can be determined by the following formula.

$$\textit{Speedup required} = \frac{\textit{Execution time without Versat}}{\textit{Real-time}} \quad (4)$$

TableIX presents the outcomes of the previously discussed calculations.

With the previous results, it is noticeable that the *spectrum_search* accelerator provides a speedup very close to the desired one. However, the required speedup indicates that the acceleration of *psycho_3_threshold* is not enough to meet the real-time requirements. The last two input files' encoding (most relevant cases) is approximately at 2.42 and 2.30 speedup from meeting the real-time requirements. These

	Input file			
	short.wav	long.wav	noise.wav	vivaldi.wav
twolame_psycho_3_init	635	642	635	635
mem → fft_buf	5	133	115	155
mem → fft_buf	6	117	77	120
psycho_3_fft	47	1138	886	1253
psycho_3_powerdensityspectrum	40	1026	791	1115
psycho_3_spl	7	170	148	204
psycho_3_tonal_label	7	153	258	173
psycho_3_noise_label	8	220	167	235
psycho_3_dump	-	4	2	4
psycho_3_decimation	3	45	38	67
psycho_3_threshold	611	15168	12975	16684
psycho_3_minimummasking	-	23	14	29
psycho_3_smr	-	9	6	13
***TwoLAME* total**	1614	24719	20210	26467

TABLE V: Execution time for all input files (third phase of *profiling*) [ms].

Metric	**without *Versat***	**with *Versat***	
		spectrum_search	*masking_threshold*
Total LUTs	24489	38913	34810
Logic LUTs	19818	33359	29267
LUTRAMs	4288	5160	5152
SRLs	383	394	391
Flip-Flops	24888	37653	35186
RAMB36	157	161	176
RAMB18	5	5	6
URAM	0	0	0
DSP Blocks	10	26	14

TABLE VI: FPGA implementation results of IOb-MP2-E with and without *Versat*.

		Input files			
		short.wav	*long.wav*	*noise.wav*	*vivaldi.wav*
without *Versat*	*psycho_3_threshold*	536	13365	11381	14694
	TwoLAME total	1509	22672	18520	24325
with ***spectrum_search***	*psycho_3_threshold*	20	350	289	395
	TwoLAME total	974	9391	7209	9729
with ***masking_threshold***	*psycho_3_threshold*	555	13807	11761	15168
	TwoLAME total	1518	22677	18637	24458

TABLE VII: Execution time of IOb-MP2-E with and without *Versat* for all input files [ms].

	Input file			
Speedup	*short.wav*	*long.wav*	*noise.wav*	*vivaldi.wav*
achieved with *spectrum_search*	1.549	2.414	2.569	2.5
desired	1.551	2.436	2.594	2.526
required	5.251	2.9321	6.219	5.748

TABLE IX: Speedup achieved, desired, and required for all input files.

results give some guarantees about achieving the goal of this work since the third input file has a considerable number of frames and represents the worst-case scenario for real-time encoding (being a stereo white noise audio file). The fourth input file has a normal audio complexity since it is part of an existing soundtrack, and so the difference between achieved and required speedup is similar to the previous file.

The speedup values give a direct relation to the frequency necessary to meet the real-time requirements. Picking the *noise.wav* input file as an example, the required speed up for IOB-MP2-E without hardware acceleration is 6.2. Since IOb-MP2-E operates at 100MHz, this means that an implementation for 620MHz would meet the goal.

VI. Conclusion

In this work, the first system RISC-V-based MP2 audio encoder has been developed to the best of our knowledge. The system has been implemented with the IOb-SoC system equipped with the *VEXRISC* CPU and the *Versat* reconfigurable accelerator.

Compared with the commercial *CWda74* IP Core, the support for floating-point arithmetic allows us to use high-precision and good-quality open-source software such as *TwoLame*, as we have not found any fixed-point implementations. The *CWda74* uses third-party proprietary software and needs to pay royalties for its use. However, due to its fixed-point performance, it uses considerably lower resources and can work at a substantially lower frequency.

A. Achievements

The most notable results of this work are summarized below.

- A new IOb-SoC system called IOb-MP2-E has been developed to support the development of the embedded MP2 audio encoder.
- The *TwoLAME*'s library was successfully ported to the IOb-MP2-E with floating-point precision.
- Software optimizations were done to reduce unnecessary operations related to mathematical functions and memory allocation.
- Using the FPGA implementation, the software was profiled to determine which functions took most of the execution time. The *psycho_3_threshold* was marked as the function to be accelerated using *Versat*.
- Two hardware accelerators were developed, each corresponding to a different part of the function.
- The FPGA implementation results demonstrate that the addition of the *Versat* accelerator reduces the encoding time considerably, with results close to the theoretical minimum. The obtained speed was not enough to meet the real-time requirements at the implementation frequency of 100MHz. The achieved performance for the worst-case scenario shows that an implementation at 240MHz would meet the goal.
- The FPGA implementation results also demonstrate that the addition of the *Versat* accelerator has a considerable impact on the resources used in a factor of 10x. It suggests that a smaller but frequently reconfigured single accelerator is preferable.

B. Future work

The work carried out demonstrates that the main area for improvement is to reduce the resources necessary. Some improvements that have been identified are outlined below.

- Make the accelerator reconfigurable to perform multiple functions while reusing its hardware and implementing various accelerators.
- Explore using custom instructions created to extend the *VEXRISCV*'s ISA, attempting to accelerate *TwoLAME* algorithm horizontally.

These findings contribute to the ongoing efforts to provide more efficient and competitive solutions for audio encoding in MPEG-1/2 Layer II format, with the potential for real-world applications in broadcasting and multimedia.

REFERENCES

[1] I. Conexant Systems, *MPEG-2 Codec CX23415*, 2003, accessed Dec 2022. [Online]. Available: https://datasheet.ciiva.com/26931/getdatasheetpartid-486546-26931824.pdf

[2] CableWorld, *MPEG-2 Encoder CW-4888*, 2014, accessed Dec 2022. [Online]. Available: http://www.cableworld.hu/downloads/data_sheets/4888p-a.pdf

[3] I. Computer Modules, *Futura II ASI+IP™*, 2017, accessed Dec 2022. [Online]. Available: https://cdn-docs.av-iq.com/dataSheet/Futura%20II%E2%84%A2%20SDI%20HDSDI%20HDMI%20-ASI%2BIP.pdf

[4] "MPEG-1/2 - Layer i/ii Audio Encoder," Oct 2017, accessed Dec 2022. [Online]. Available: http://coreworks-sa.com/index.php?view=view_ip_core&part_number=CWda74&ipcore_id=57&category=Audio%20Encoders

[5] L. IPbloq, *MPEG-1/2 - LAYER I/II AUDIO ENCODER*, 2019, accessed Dec 2022. [Online]. Available: https://ipbloq.files.wordpress.com/2021/09/ipb-mpeg-se-product-brief.pdf

[6] IOBundle, "IOBundle Website," 2023, accessed Sep 2023. [Online]. Available: https://www.iobundle.com/

[7] IObundle, "IObundle/iob-soc: IOB SoC Repository," 2023, accessed Apr 2023. [Online]. Available: https://github.com/IObundle/iob-soc

[8] E. A. Waterman and K. Asanović, *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20191213*, RISC-V Foundation, December 2019, accessed 2023.

[9] njh, "njh/twolame: TwoLAME is an optimised MPEG Audio Layer 2 (MP2) encoder," 2022, accessed 2023. [Online]. Available: https://github.com/njh/twolame

[10] I. O. for Standardization, "Iso 22412:2020," International Organization for Standardization (ISO), 2022, accessed Dec 2022. [Online]. Available: https://www.iso.org/standard/22412.html

[11] N. Humfrey, "TwoLAME - An Optimized MPEG Audio Layer 2 (MP2) Encoder," 2022, accessed Jul 2023. [Online]. Available: https://www.twolame.org/

[12] U. Farooq, Z. Marrakchi, and H. Mehrez, *FPGA Architectures: An Overview*. New York, NY: Springer New York, 2012, pp. 7–48, accessed Oct 2023. [Online]. Available: https://doi.org/10.1007/978-1-4614-3594-5_2

[13] IObundle, "IObundle/iob-versat: Versat Repository," 2023-03, accessed Aug 2023. [Online]. Available: https://github.com/IObundle/iob-versat

[14] Y. Wang and M. Vilermo, "Modified discrete cosine transform: Its implications for audio coding and error concealment," *Journal of the Audio Engineering Society*, vol. 51, no. 1/2, pp. 52–61, 2003, accessed 2023.

[15] N. Einspruch, *Application specific integrated circuit (ASIC) technology*. Academic Press, 2012, vol. 23, accessed 2022.

[16] IObundle, "IOB SoC GitHub Repository," https://github.com/IObundle/iob-soc, accessed December 2022.

[17] N. Rathi, A. Kumar, N. Gupta, and S. K. Singh, "A review of low-power static random access memory (sram) designs," in *2023 IEEE Devices for Integrated Circuit (DevIC)*, 2023, pp. 455–459, accessed 2023.

[18] J. D. Lopes, M. P. Véstias, R. P. Duarte, H. C. Neto, and J. T. de Sousa, "Coarse-grained reconfigurable computing with the versat architecture," *Electronics*, vol. 10, no. 6, 2021, acessed Oct 2023. [Online]. Available: https://www.mdpi.com/2079-9292/10/6/669

[19] J. Liang, R. Tessier, and O. Mencer, "Floating point unit generation and evaluation for fpgas," in *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003. FCCM 2003.*, 2003, pp. 185–194, acessed 2023.

[20] Xilinx, *AXI Reference Guide*, March 2011, acessed Aug 2023. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf

[21] GNU Project, "Linux and the GNU System," 2023, acessed Sep 2023. [Online]. Available: https://www.gnu.org/gnu/linux-and-gnu.en.html

[22] M. G. . contributors., "Audacity FAQ," Audacity Team, 2023, acessed 2023. [Online]. Available: https://www.audacityteam.org/FAQ/

[23] cppreference.com, "modf, modff, modfl," 2023, acessed March 2023. [Online]. Available: https://en.cppreference.com/w/c/numeric/math/modf

[24] TutorialsPoint, "C library - <math.h>," 2023, acessed Mar 2023. [Online]. Available: https://www.tutorialspoint.com/c_standard_library/math_h.htm

[25] R. Patel, "A survey of embedded software profiling methodologies," *International Journal of Embedded Systems and Applications*, vol. 1, no. 2, pp. 19–40, dec 2011, acessed May 2023. [Online]. Available: https://doi.org/10.5121%2Fijesa.2011.1203

[26] L. FMALL Co., "XCKU040-1FBVA676I FPGA Product," 2023, acessed Oct 2023. [Online]. Available: https://www.fmall.uk/product/xilinx/xcku040-1fbva676i

[27] G. M. Amdahl, "Computer architecture and amdahl's law," *Computer*, vol. 46, no. 12, pp. 38–46, dec 2013, acessed 2023.