VHDLwhiz.com

WAV audio file reader/writer packages

Version:	1.0.0
Date:	July 1, 2022
Product URL:	https://vhdlwhiz.com/product/vhdl-package-wav-audio-file-reader-writer/
Contact email:	jonas@vhdlwhiz.com

This document describes how to use the *wav_reader* and *wav_writer* VHDL packages for reading and writing uncompressed WAV audio format files.

Table of content

License

The MIT license covers the source code’s copyright requirements and terms of use. Refer to the *LICENSE.txt* file in the Zip file for details.

Changelog

These changes refer to the project files, and this document is updated accordingly.

Version	**Remarks**
1.0.0	Initial release

Description

This project contains two VHDL packages for reading from or writing audio data to WAV files in simulation.

WAV is a file format for storing audio data that's widely used on Windows-based systems and by audio recording devices or editing software like the Adobe suite. Despite the emergence of more efficient audio codec algorithms, WAV remains a popular format for storing uncompressed, lossless audio data because of its simplicity and cross-platform support.

Every WAV file begins with a Resource Interchange File Format (RIFF) header containing information about the audio data's encoding. After this mandatory part, there may be multiple subchunks with meta info that's not crucial for knowing how to interpret the audio data.

Therefore, VHDLwhiz's *wav_reader* package ignores non-mandatory parts and skips to the "data" subchunk containing the audio samples. Similarly, the *wav_writer* package can reproduce or create a new WAV file with a mandatory RIFF header.

The VHDL packages support WAV files that are mono or stereo, have any sampling rate, and any sample bit length, as long as it's a multiple of 8 bits.

WAV files support any type of encoding on the individual audio samples, but the most common ones are pulse-code modulation (PCM) or 32-bit IEEE floating-point.

The *wav_reader* package allows you to extract the format code from the file header, but it will not do any conversion. It reads the audio samples as-is, and it's up to the user of the package to interpret the data. Likewise, you may specify the format code when using the *wav_writer* package to indicate the encoding of the samples.

Example use cases

The comments above the method prototype code in the VHDL files describe how the procedures and functions in the protected types work.

Study the VHDL testbench to see a working example that reads and writes mono and stereo WAV files with PCM and IEEE float encoding.

DUT audio passthrough

When testing audio processing modules, a typical use case is to read from one WAV file, stream the data through the device under test (DUT), and write back the results to a new WAV file.

That's easy to accomplish with VHDLwhiz's WAV file reader/writer packages.

The example code below shows how you can set up such a testbench.

```
READER_PROC : process
begin
  -- Open WAV file for reading
  reader.open_file(read_filename);
  reader.print_metainfo;

  -- Open WAV file for writing with the same header fields
  writer.open_file(
    write_filename,
    reader.get_audio_format,
    reader.get_num_channels,
    reader.get_sample_rate,
    reader.get_bits_per_sample,
    reader.get_total_samples);

  while not reader.is_empty loop

    -- Send data to the DUT
    reader.read_sample(stereo_in_l, stereo_in_r);

    -- Wait until the next sample while the DUT is processing
    wait for 1 sec / reader.get_sample_rate;

    -- Write the result to the output file
    writer.write_sample(stereo_out_l, stereo_out_r);

  end loop;

  -- Close files
  reader.close_file;
  writer.close_file;
  wait;
end process;
```

First, we open the input WAV file for reading. Then, we create an output WAV file based on the input file's header. After that, we enter a loop that reads each sample and sends it to the DUT.

Because audio data is slow compared to the FPGA's high clock speed, the DUT will have plenty of time to process the samples before we write its output to the second WAV file.

Finally, we close both WAV files so that we can listen to them or analyze them using third-party software.

Method prototypes

wav_reader_pkg.vhd

The code listing below shows the declarative region of the WAV file reader package.

```
package wav_reader_pkg is
  type wav_reader is protected

    -- Open the wave file for reading
    --
    -- @param filename The path to the input file
    --
    procedure open_file(filename : string);

    -- Close the wave file if open
    procedure close_file;

    -- Check if the file has been opened for reading
    --
    -- @return true if the file is open
    --
```

```vhdl
    impure function is_open return boolean;

    -- Check if more data is available for reading from the file
    impure function is_empty return boolean;

    -- Code identifying the encoding of each sample channel
    -- 1 = Pulse-code modulation (PCM)
    -- 2 = Adaptive differential pulse-code modulation (ADPCM)
    -- 3 = IEEE floating-point number
    -- Etc.
    impure function get_audio_format return integer;

    -- The number of audio channels (mono = 1, stereo = 2)
    impure function get_num_channels return integer;

    -- The sampling rate in Hz
    impure function get_sample_rate return integer;

    -- Bits per sample, per channel
    impure function get_bits_per_sample return integer;

    -- The total number of samples in this file (sample sets, not channels)
    impure function get_total_samples return integer;

    -- The number of unread samples since opening this file
    impure function get_unread_samples return integer;

    -- Print the wave file header and other information
    procedure print_metainfo;

    -- Read audio sample (mono version)
    procedure read_sample(signal sample : out std_logic_vector);

    -- Read audio sample (stereo version)
    procedure read_sample(signal sample_l, sample_r : out std_logic_vector);

  end protected;
end package;
```

wav_writer_pkg.vhd

The code listing below shows the declarative region of the WAV file writer package.

```
package wav_writer_pkg is
  type wav_writer is protected

    -- Open the wave file for writing
    --
    -- In addition to the filename, you need to supply information that goes into
    -- the header of the output wave file.  If you are writing back samples that
    -- initially came from the wav_reader_pkg package, you can access those data
    -- from the reader's getter functions: get_audio_format, get_num_channels,
    -- get_sample_rate, get_bits_per_sample, get_total_samples,
and  get_unread_samples.
    --
    -- @param filename The path to the input file
    -- @param audio_format Numeric value identifying the encoding of a sample
channel
    -- @param num_channels Mono = 1, stereo = 2
    -- @param sample_rate The sampling rate in Hz
    -- @param bits_per_sample Bits per sample, per channel (must be a multiple of
8)
    -- @param total_samples The exact number of samples you intend to write to
this file
    --
    procedure open_file(
      filename : string;
      audio_format : integer;
      num_channels : integer;
      sample_rate : integer;
      bits_per_sample : integer;
      total_samples : integer);

    -- Close the wave file if open
    --
    -- You must call this procedure after calling write_sample()
    -- total_samples number of times.
    --
    procedure close_file;

    -- Check if the file has been opened for writing
    --
    -- @return true if the file is open
    --
    impure function is_open return boolean;

    -- The number of samples left to write based on total_samples
    impure function get_samples_left_to_write return integer;

    -- Write one audio sample (mono version)
    procedure write_sample(constant sample : in std_logic_vector);

    -- Write one audio sample set (stereo version)
    procedure write_sample(constant sample_l, sample_r : in std_logic_vector);

  end protected;
end package;
```

Zip file content

.	
├── **WAV file RW package - User Manual.pdf**	This document
├── **wav_reader_pkg.vhd**	The VHDL package for reading WAV files
├── **wav_writer_pkg.vhd**	The VHDL package for writing WAV files
├── **testbench.vhd**	VHDL testbench for the package
├── **How to run.gif**	Screencast guide for running the testbench
├── **LICENSE.txt**	License agreement
├── **project.mpf**	ModelSim/Questa project file
├── **run.do**	ModelSim/Questa script for running the testbench
├── **wave.do**	Wave format file for ModelSim/Questa
├── **Audio_in_mono_48kHz_32b_float.wav**	Audio sample
└── **Audio_in_stereo_48kHz_24b_PCM.wav**	Audio sample

Simulating the design

There is a demo testbench in the Zip file (*testbench.vhd*).

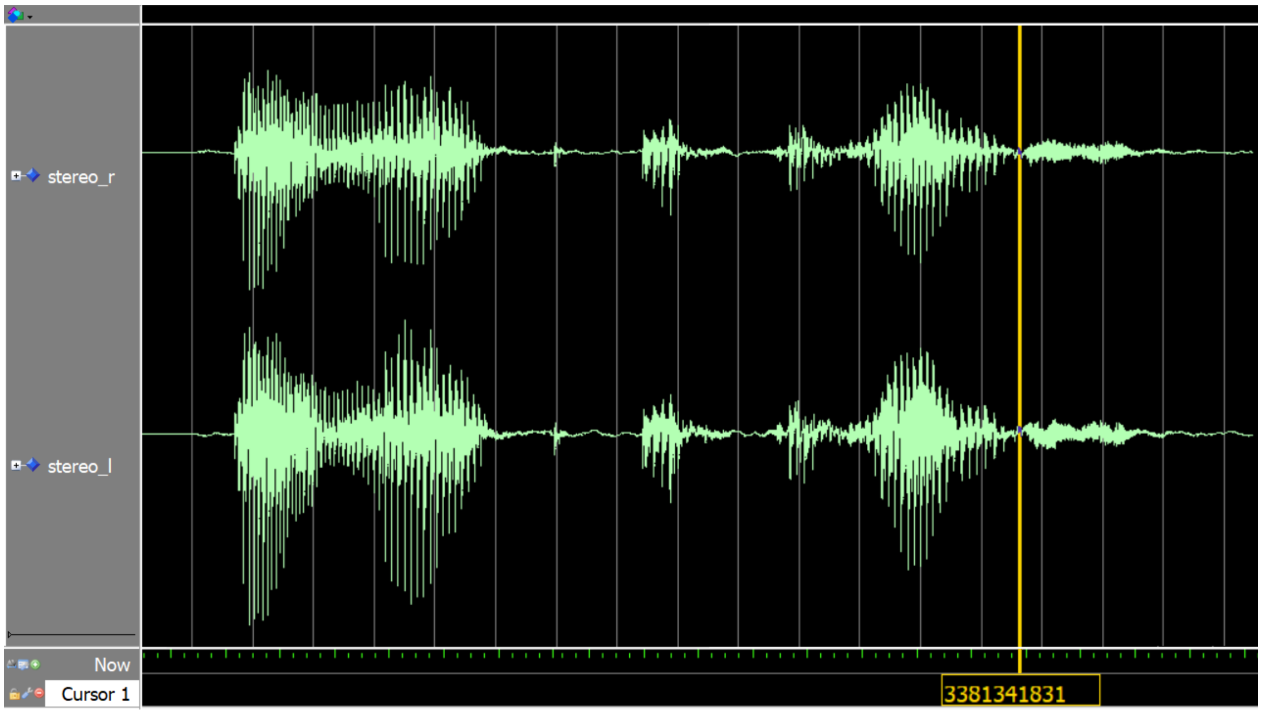
The VHDL testbench should work in any capable VHDL simulator supporting the full 2008 VHDL revision, but the provided *run.do* script only works in ModelSim/Questa.

To run the testbench, open ModelSim/Questa and type in the simulator console:

```
do <path_to_extracted_zip_content>/run.do
runtb
```

The testbench reads two WAV files using the *wav_reader* and writes them back to disk using the *wav_writer* package. The two input files are **Audio_in_mono_48kHz_32b_float.wav** and **Audio_in_stereo_48kHz_24b_PCM.wav**. The output files will be **Audio_out_mono.wav** and **Audio_out_stereo.wav**.

The WAV header information should appear in the simulator's transcript window, and a waveform showing the audio as an analog signal should appear, as shown below.



Known issues

The package is unsynthesizable and only meant for use in testbenches/simulation.