VHDLwhiz.com

Binary file reader/writer packages

Version:	1.0.2
Date:	July 11, 2022
Product URL:	https://vhdlwhiz.com/product/vhdl-package-binary-file-reader-writer/
Contact email:	jonas@vhdlwhiz.com

This document describes how to use VHDLwhiz's binary file reader and writer VHDL simulation packages that handle vectors of any length.

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
1.0.1	Corrected method prototype comments. No functional changes.
1.0.2	Adding a read_bits function as an alternative to the procedure

Description

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

The new packages' protected types (VHDL classes) provide a more straightforward interface for file access than VHDL's standard TEXTIO package. Additionally, it allows for uncomplicated reading and writing of data chunks with arbitrary bit lengths.

It's useful for testbenches operating on binary files where the information fields don't fit into byte boundaries, for example, image files or network packages.

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.

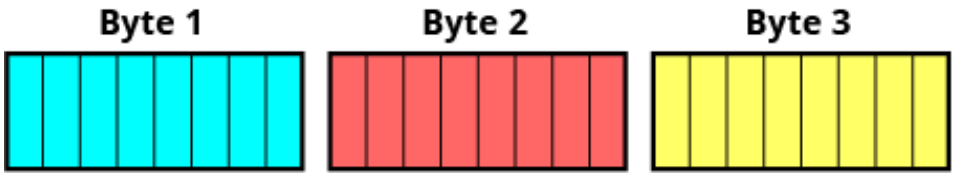
Study the VHDL testbench to see a working example that writes and reads back data from binary files, and review the examples below to see how writing vectors of different lengths affect the stored bytes in files.

Writing single bytes

When writing 8-bit chunks through the writer, they are stored as discrete bytes in the output file, as shown below.

Each written 8-bit value perfectly occupies a byte on the filesystem, and this is also straightforward to accomplish with the standard TEXTIO library.

```
writer.write_bits("11111111");
                  writer.write_bits("11111111");
                                    writer.write_bits("11111111");
```

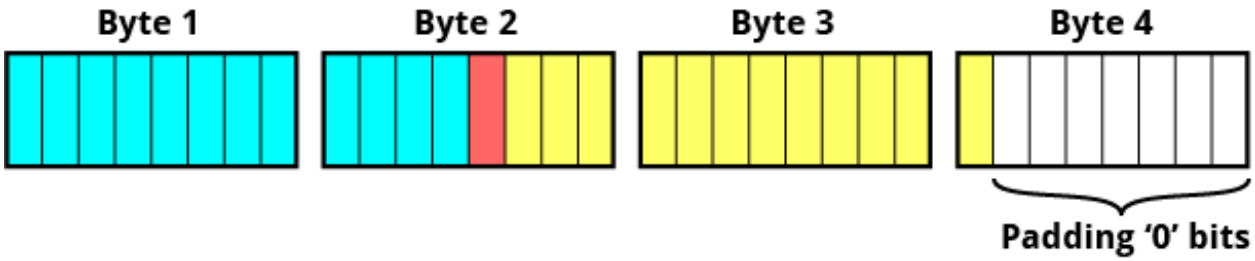


Writing vectors of different lengths

On the other hand, writing different length vectors to the file is cumbersome with TEXTIO, but no different from writing bytes with VHDLwhiz's packages.

In the example below, we first write a 12-bit vector, then a single bit, and finally, a 12-bit vector again to the same file. The illustration below shows how the information is stored in the output file.

```
writer.write_bits("111111111111");
                  writer.write_bits("1");
                                    writer.write_bits("111111111111");
```



Note that the last byte is zero-padded because the writes haven't filled it.

Reading is the exact opposite operation which reconstructs the data and discards any padding bits.

Method prototypes

file_reader_pkg.vhd

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

```
package file_reader_pkg is
  type file_reader is protected

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

    -- Close the binary file if open
    procedure close_file;

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

    -- Check if more data is available for reading from the file
    --
    -- @param min_bits_left The number of bits in the next potential read.
    --    Values > 8 default to 8 (one byte).
    --    The function has to know if the next expected read size is less than
    --    one byte to determine if the remaining bits are padding or data bits.
    --
    -- @return true if there are at least'min_bits_left' bits left to read
    --
    impure function is_empty(min_bits_left : integer := 8) return boolean;

    -- Read from the input file
    --
    -- @param n_bits The number of bits to read from the file
    --
    -- @return A vector containing the read bits
    --
    impure function read_bits(n_bits : integer) return std_logic_vector;

    -- Read from the input file.
    -- This procedure does the same as the read_bits function,
```

```
    -- but the procedure only works when assigning to a variable.
    --
    -- @param bits The vector to put the read bits into.
    --    The procedure will read bits'length number of bits from the file.
    --
    procedure read_bits(variable bits : out std_logic_vector);

  end protected;
end package;
```

file_writer_pkg.vhd

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

```
package file_writer_pkg is
  type file_writer is protected

    -- Open the binary file for writing
    --
    -- If the file exists, it will be overwritten.
    --
    -- @param filename The path to the input file
    --
    procedure open_file(filename : string);

    -- Close the output file if open
    --
    -- Before exiting the simulation,
    -- you must call this method to ensure that every bit gets written.
    --
    -- If the total number of written bits is not a multiple of 8,
    -- the output file will be padded with zero bits in the final byte.
    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;

    -- Write data to the output file
    --
    -- @param bits The vector containing the data to write.
    --    Every bit from the bits vector will be written to file.
```

```
    --
    procedure write_bits(bits : in std_logic_vector);

  end protected;
end package;
```

Zip File Content

.

├── **Binary file RW package - User Manual.pdf**	This document
├── **file_reader_pkg.vhd**	The VHDL package for reading binary files
├── **file_writer_pkg.vhd**	The VHDL package for writing binary files
├── **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
└── **testbench.vhd**	Self-checking VHDL testbench for the packages

Simulating the design

There is a self-checking 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
```

Known Issues

Endianness may vary between operating systems, simulators, and implementation tools. However, the reader package should always be able to reconstruct the data created with the writer package on the same system setup.

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