

VPI_BINARY_FILE_IO



November 26, 2024

Jay Convertino

Contents

| | |
|---|----------|
| 1 Usage | 2 |
| 1.1 Introduction | 2 |
| 1.2 Dependencies | 3 |
| 1.2.1 fusesoc_info Depenecies | 3 |
| 2 Architecture | 3 |
| 3 Building | 4 |
| 3.1 fusesoc | 4 |
| 3.2 Source Files | 4 |
| 3.2.1 fusesoc_info File List | 4 |
| 3.3 Targets | 5 |
| 3.3.1 fusesoc_info Targets | 5 |
| 3.4 Directory Guide | 5 |
| 4 Simulation | 6 |
| 5 Code Documentation | 7 |
| 5.1 vpi binary file code document | 8 |

1 Usage

1.1 Introduction

This library provides two functions.

- `read_binary_file(FILE_NAME, VECTOR)`
- `write_binary_file(FILE_NAME, VECTOR)`

Each instance is a new instance, and will start reading the file from the start. The vector has to be in size bytes from 1 to any number of bytes. Each function returns the number of bytes read or written. Z or X place in the vector indicates bytes not available for read, or do not write these bytes for write. The read function will return a negative number of bytes when the end of file is reached.

You can use the following for including the library in your project:

```
src :
  files :
    - src/read_binary_file.c : {file_type : cSource}
    - src/write_binary_file.c : {file_type : cSource}
    - src/binary_file_io.c   : {file_type : cSource}
    - src/binary_file_io.h   : {file_type : cSource,
      ↪ is_include_file : true}
    - src/read_binary_file.h : {file_type : cSource,
      ↪ is_include_file : true}
    - src/write_binary_file.h : {file_type : cSource,
      ↪ is_include_file : true}
    - src/binary_file_io.sft : {file_type : user}

lib :
  files :
    - lib_ringbuffer/build/libringBuffer.a : {file_type :
      ↪ user, copyto : .}

header :
  files :
    - lib_ringbuffer/ringBuffer.h : {file_type : cSource,
      ↪ is_include_file : true}

vpi:
  bin_file_io_vpi:
    filesets : [src, header]
    libs : [ringBuffer -L., pthread]

generate:
```

```

gen_git:
  generator: git_pull
  parameters:
    repo_url: https://github.com/sparkletron/
    ↪ C89_pthread_ring_buffer.git
    repo_dir: lib_ringbuffer
    tag: release_v1.6.1
gen_lib:
  generator: gen_cmake
  parameters:
    src_dir: lib_ringbuffer
    cmake_args: ["-DCMAKE_POSITION_INDEPENDENT_CODE=ON"
    ↪ ]

targets:
  default: &default
  description: Intergration default target for
    ↪ simulations.
  filesets: [lib, dep_gen]
  generate: [gen_git, gen_lib]
  vpi: [bin_file_io_vpi]

```

1.2 Dependencies

The following are the dependencies of the cores.

- fusesoc 2.X
- iverilog (simulation)

1.2.1 fusesoc_info Depenecies

- dep_tb
 - AFRL:utility:sim_helper
- dep_gen
 - AFRL:utility:generators:1.0.0

2 Architecture

This VPI library provides two functions for the user to use during simulation, `read_binary_file` and `write_binary_file`. These will read and

write from binary files. These functions use ringbuffers and multi-threading to separate file I/O from the simulation so file access will not slow down the simulation.

The `read_binary_file` will read any binary file till it runs out of data. When it does, if it can not complete the word (one byte left, for say a 4 byte word output) then the unused bytes for the `aval/bval` pairs are set to Z. Meaning in the simulation they will show up as Z, not 0 or 1. It will also assert the EOF (end of file) signal from the core showing that this is the last of the data.

The `write_binary_file` will write any binary data till it is given that is a 0 or a 1. Any bytes that contain a Z will not be written to the output file. This allows for any file that is read to be written in a one to one manner.

Please see 5 for more information per target.

3 Building

The all VPI binary file IO source files are written in C to target the VPI API from Verilog 2001. They should simulate in any modern simulation tool that has VPI support. The library comes as a `fusesoc` packaged core and can be included in any other testbench. Be sure to make sure you have met the dependencies listed in the previous section.

3.1 fusesoc

Fusesoc is a system for building FPGA software without relying on the internal project management of the tool. Avoiding vendor lock in to Vivado or Quartus. These cores, when included in a project, can be easily integrated and targets created based upon the end developer needs. The core by itself is not a part of a system and should be integrated into a `fusesoc` based system. Simulations are setup to use `fusesoc` and are a part of its targets.

3.2 Source Files

3.2.1 fusesoc_info File List

- `src`
 - `'src/read_binary_file.c': 'file_type': 'cSource'`
 - `'src/write_binary_file.c': 'file_type': 'cSource'`
 - `'src/binary_file_io.c': 'file_type': 'cSource'`
 - `'src/binary_file_io.h': 'file_type': 'cSource', 'is_include_file': True`

- 'src/read_binary_file.h': 'file_type': 'cSource', 'is_include_file': True
- 'src/write_binary_file.h': 'file_type': 'cSource', 'is_include_file': True
- 'src/binary_file_io.sft': 'file_type': 'user'
- lib
 - 'lib_ringbuffer/build/libringBuffer.a': 'file_type': 'user', 'copyto': ''
- header
 - 'lib_ringbuffer/ringBuffer.h': 'file_type': 'cSource', 'is_include_file': True
- tb
 - 'tb/tb_vpi.v': 'file_type': 'verilogSource'

3.3 Targets

3.3.1 fusesoc_info Targets

- default

Info: Intergration default target for simulations.
- sim

Info: Test VPI file io.
- sim_rand_data

Info: Test VPI file io with random data.
- sim_8bit_count_data

Info: Test VPI file io with count data.

3.4 Directory Guide

Below highlights important folders from the root of the directory.

1. **docs** Contains all documentation related to this project.
 - **manual** Contains user manual and github page that are generated from the latex sources.
2. **src** Contains source files for vpi binary file io.
3. **tb** Contains test bench files.

4 Simulation

A barebones test bench for iverilog is included in `tb/tb_vpi.v` . This can be run from `fusesoc` with the following.

```
$ fusesoc run --target=sim AFRL:vpi:binary_file_io:1.0.0
```

5 Code Documentation

- **VPI BINARY FILE SOURCE, DOXYGEN**

The next section documents the library.

VPI_BINARY_FILE_IO

1.0

Generated by Doxygen 1.9.1

| | |
|---------------------------------------|----------|
| 1 Data Structure Documentation | 1 |
| 1.1 s_process_data Struct Reference | 1 |
| 1.1.1 Field Documentation | 1 |
| 1.1.1.1 arg2_handle | 1 |
| 1.1.1.2 array_byte_size | 1 |
| 1.1.1.3 error | 2 |
| 1.1.1.4 num_ab_val_pairs | 2 |
| 1.1.1.5 p_file | 2 |
| 1.1.1.6 p_file_name | 2 |
| 1.1.1.7 p_ringbuffer | 2 |
| 1.1.1.8 systf_handle | 2 |
| 1.1.1.9 thread | 2 |
| 2 File Documentation | 3 |
| 2.1 binary_file_io.c File Reference | 3 |
| 2.1.1 Function Documentation | 4 |
| 2.1.1.1 binary_compiletf() | 4 |
| 2.1.1.2 binary_end_compile_cb() | 4 |
| 2.1.1.3 binary_end_sim_cb() | 4 |
| 2.1.1.4 binary_sizetf() | 4 |
| 2.1.1.5 read_binary_reg_systf() | 5 |
| 2.1.1.6 write_binary_reg_systf() | 5 |
| 2.1.2 Variable Documentation | 5 |
| 2.1.2.1 vlog_startup_routines | 5 |
| 2.2 binary_file_io.h File Reference | 5 |
| 2.2.1 Detailed Description | 6 |
| 2.2.2 Macro Definition Documentation | 7 |
| 2.2.2.1 BUFFSIZE | 7 |
| 2.2.2.2 DATACHUNK | 7 |
| 2.2.2.3 READ_NAME | 7 |
| 2.2.2.4 WRITE_NAME | 7 |
| 2.2.3 Function Documentation | 8 |
| 2.2.3.1 binary_end_compile_cb() | 8 |
| 2.2.3.2 binary_end_sim_cb() | 8 |
| 2.2.3.3 binary_sizetf() | 8 |
| 2.3 read_binary_file.c File Reference | 8 |
| 2.3.1 Detailed Description | 9 |
| 2.3.2 Function Documentation | 9 |
| 2.3.2.1 read_binary_calltf() | 9 |
| 2.3.2.2 read_binary_start_sim_cb() | 10 |
| 2.3.2.3 read_thread() | 10 |
| 2.4 read_binary_file.h File Reference | 10 |

| | |
|--|-----------|
| 2.4.1 Detailed Description | 11 |
| 2.4.2 Function Documentation | 11 |
| 2.4.2.1 read_binary_calltf() | 11 |
| 2.4.2.2 read_binary_start_sim_cb() | 12 |
| 2.5 write_binary_file.c File Reference | 12 |
| 2.5.1 Detailed Description | 12 |
| 2.5.2 Function Documentation | 13 |
| 2.5.2.1 write_binary_calltf() | 13 |
| 2.5.2.2 write_binary_start_sim_cb() | 13 |
| 2.5.2.3 write_thread() | 13 |
| 2.6 write_binary_file.h File Reference | 14 |
| 2.6.1 Detailed Description | 14 |
| 2.6.2 Function Documentation | 15 |
| 2.6.2.1 write_binary_calltf() | 15 |
| 2.6.2.2 write_binary_start_sim_cb() | 15 |
| Index | 17 |

Chapter 1

Data Structure Documentation

1.1 s_process_data Struct Reference

```
#include <binary_file_io.h>
```

Data Fields

- `PLI_INT32` [error](#)
- `PLI_INT32` [num_ab_val_pairs](#)
- `PLI_INT32` [array_byte_size](#)
- `char *` [p_file_name](#)
- `struct s_ringBuffer *` [p_ringbuffer](#)
- `FILE *` [p_file](#)
- `pthread_t` [thread](#)
- `vpiHandle` [systf_handle](#)
- `vpiHandle` [arg2_handle](#)

1.1.1 Field Documentation

1.1.1.1 arg2_handle

```
vpiHandle s_process_data::arg2_handle
```

1.1.1.2 array_byte_size

```
PLI_INT32 s_process_data::array_byte_size
```

1.1.1.3 error

```
PLI_INT32 s_process_data::error
```

1.1.1.4 num_ab_val_pairs

```
PLI_INT32 s_process_data::num_ab_val_pairs
```

1.1.1.5 p_file

```
FILE* s_process_data::p_file
```

1.1.1.6 p_file_name

```
char* s_process_data::p_file_name
```

1.1.1.7 p_ringbuffer

```
struct s_ringBuffer* s_process_data::p_ringbuffer
```

1.1.1.8 systf_handle

```
vpiHandle s_process_data::systf_handle
```

1.1.1.9 thread

```
pthread_t s_process_data::thread
```

The documentation for this struct was generated from the following file:

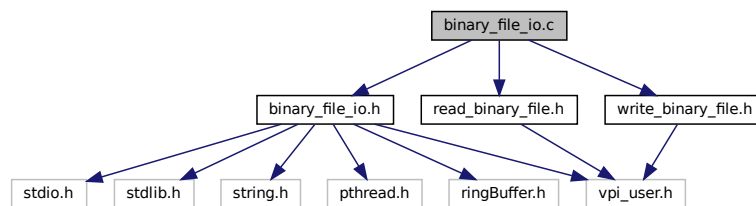
- [binary_file_io.h](#)

Chapter 2

File Documentation

2.1 `binary_file_io.c` File Reference

```
#include "binary_file_io.h"  
#include "read_binary_file.h"  
#include "write_binary_file.h"  
Include dependency graph for binary_file_io.c:
```



Functions

- `PLI_INT32 binary_end_compile_cb` (`p_cb_data data`)
BINARY FILE END COMPILE CALLBACK.
- `PLI_INT32 binary_end_sim_cb` (`p_cb_data data`)
BINARY FILE END SIM CALLBACK.
- `PLI_INT32 binary_sizetf` (`PLI_BYTE8 *user_data`)
Returns the size, in bits, of the function return type.
- `PLI_INT32 binary_compiletf` (`PLI_BYTE8 *user_data`)
Compile time call, check the arguments for validity.
- `void read_binary_reg_systf` (`void`)
Setup read_binary_file function.
- `void write_binary_reg_systf` (`void`)
Setup write_binary_file function.

Variables

- void(* [vlog_startup_routines](#) [])(void)
register the new file functions

2.1.1 Function Documentation

2.1.1.1 `binary_compiletf()`

```
PLI_INT32 binary_compiletf (  
    PLI_BYTE8 * user_data )
```

Compile time call, check the arguments for validity.

2.1.1.2 `binary_end_compile_cb()`

```
PLI_INT32 binary_end_compile_cb (  
    p_cb_data data )
```

BINARY FILE END COMPILE CALLBACK.

2.1.1.3 `binary_end_sim_cb()`

```
PLI_INT32 binary_end_sim_cb (  
    p_cb_data data )
```

BINARY FILE END SIM CALLBACK.

2.1.1.4 `binary_sizetf()`

```
PLI_INT32 binary_sizetf (  
    PLI_BYTE8 * user_data )
```

Returns the size, in bits, of the function return type.

2.1.1.5 read_binary_reg_systf()

```
void read_binary_reg_systf (
    void )
```

Setup read_binary_file function.

2.1.1.6 write_binary_reg_systf()

```
void write_binary_reg_systf (
    void )
```

Setup write_binary_file function.

2.1.2 Variable Documentation

2.1.2.1 vlog_startup_routines

```
void(* vlog_startup_routines[])(void) (
    void )
```

Initial value:

```
= {
    read_binary_reg_systf,
    write_binary_reg_systf,
    0
}
```

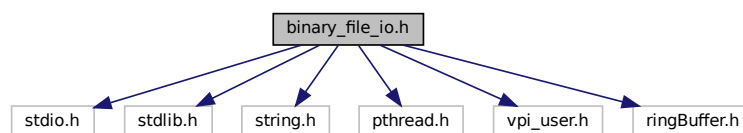
register the new file functions

2.2 binary_file_io.h File Reference

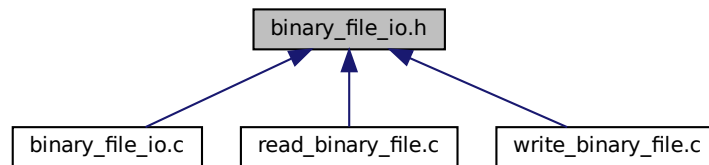
Functions to write raw binary files properly in verilog.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <vpi_user.h>
#include "ringBuffer.h"
```

Include dependency graph for binary_file_io.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [s_process_data](#)

Macros

- #define [BUFFSIZE](#) (1 << 23)
- #define [DATACHUNK](#) (1 << 21)
- #define [READ_NAME](#) "\$read_binary_file"
- #define [WRITE_NAME](#) "\$write_binary_file"

Functions

- PLI_INT32 [binary_end_compile_cb](#) (p_cb_data data)
BINARY FILE END COMPILE CALLBACK.
- PLI_INT32 [binary_end_sim_cb](#) (p_cb_data data)
BINARY FILE END SIM CALLBACK.
- PLI_INT32 [binary_size_tf](#) (PLI_BYTE8 *user_data)
Returns the size, in bits, of the function return type.

2.2.1 Detailed Description

Functions to write raw binary files properly in verilog.

Author

Jay Convertino(johnathan.convertino.1@us.af.mil)

Date

2023-20-1

@LICENSE MIT Copyright 2023 Jay Convertino

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2.2.2 Macro Definition Documentation

2.2.2.1 BUFSIZE

```
#define BUFSIZE (1 << 23)
```

2.2.2.2 DATAHUNK

```
#define DATAHUNK (1 << 21)
```

2.2.2.3 READ_NAME

```
#define READ_NAME "$read_binary_file"
```

2.2.2.4 WRITE_NAME

```
#define WRITE_NAME "$write_binary_file"
```

2.2.3 Function Documentation

2.2.3.1 `binary_end_compile_cb()`

```
PLI_INT32 binary_end_compile_cb (
    p_cb_data data )
```

BINARY FILE END COMPILE CALLBACK.

2.2.3.2 `binary_end_sim_cb()`

```
PLI_INT32 binary_end_sim_cb (
    p_cb_data data )
```

BINARY FILE END SIM CALLBACK.

2.2.3.3 `binary_sizetf()`

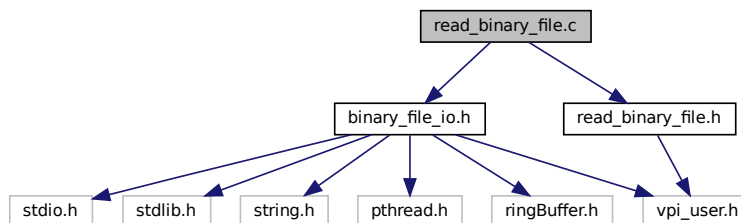
```
PLI_INT32 binary_sizetf (
    PLI_BYTE8 * user_data )
```

Returns the size, in bits, of the function return type.

2.3 `read_binary_file.c` File Reference

Functions to read raw binary files properly in verilog.

```
#include "binary_file_io.h"
#include "read_binary_file.h"
Include dependency graph for read_binary_file.c:
```



Functions

- void * `read_thread` (void *data)
READ BINARY FILE THREAD TO FILL RINGBUFFER.
- PLI_INT32 `read_binary_start_sim_cb` (p_cb_data data)
READ BINARY FILE START SIM CALLBACK.
- PLI_INT32 `read_binary_calltf` (PLI_BYTE8 *user_data)
Called by the simulator, each time it is requested.

2.3.1 Detailed Description

Functions to read raw binary files properly in verilog.

Author

Jay Convertino(johnathan.convertino.1@us.af.mil)

Date

2022-12-19

@LICENSE MIT Copyright 2023 Jay Convertino

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2.3.2 Function Documentation

2.3.2.1 read_binary_calltf()

```
PLI_INT32 read_binary_calltf (  
    PLI_BYTE8 * user_data )
```

Called by the simulator, each time it is requested.

`read_binary_calltf` is a callback for the `read_binary_file` function.

2.3.2.2 read_binary_start_sim_cb()

```
PLI_INT32 read_binary_start_sim_cb (  
    p_cb_data data )
```

READ BINARY FILE START SIM CALLBACK.

2.3.2.3 read_thread()

```
void* read_thread (  
    void * data )
```

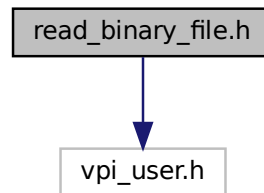
READ BINARY FILE THREAD TO FILL RINGBUFFER.

2.4 read_binary_file.h File Reference

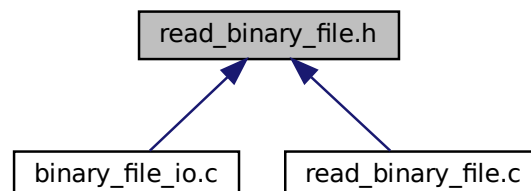
Functions to write raw binary files properly in verilog.

```
#include <vpi_user.h>
```

Include dependency graph for read_binary_file.h:



This graph shows which files directly or indirectly include this file:



Functions

- PLI_INT32 [read_binary_start_sim_cb](#) (p_cb_data data)
READ BINARY FILE START SIM CALLBACK.
- PLI_INT32 [read_binary_calltf](#) (PLI_BYTE8 *user_data)
read_binary_calltf is a callback for the read_binary_file function.

2.4.1 Detailed Description

Functions to write raw binary files properly in verilog.

Author

Jay Convertino(johnathan.convertino.1@us.af.mil)

Date

2023-20-1

\$read_binary_file takes 2 arguments. First the file name, next a register for data in size bytes. The function returns the number of bytes read. If it is a negative number, that indicates EOF.

@LICENSE MIT Copyright 2023 Jay Convertino

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2.4.2 Function Documentation

2.4.2.1 read_binary_calltf()

```
PLI_INT32 read_binary_calltf (
    PLI_BYTE8 * user_data )
```

read_binary_calltf is a callback for the read_binary_file function.

read_binary_calltf is a callback for the read_binary_file function.

2.4.2.2 read_binary_start_sim_cb()

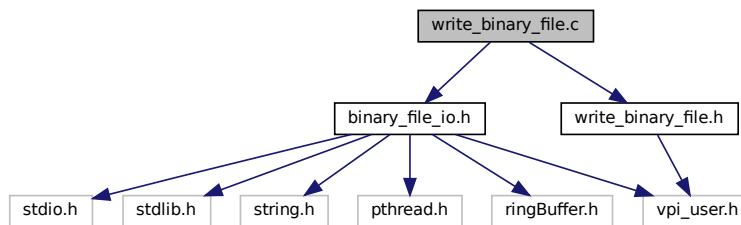
```
PLI_INT32 read_binary_start_sim_cb (
    p_cb_data data )
```

READ BINARY FILE START SIM CALLBACK.

2.5 write_binary_file.c File Reference

Functions to write raw binary files properly in verilog.

```
#include "binary_file_io.h"
#include "write_binary_file.h"
Include dependency graph for write_binary_file.c:
```



Functions

- void * [write_thread](#) (void *data)
WRITE BINARY FILE THREAD TO EMPTY RINGBUFFER.
- PLI_INT32 [write_binary_start_sim_cb](#) (p_cb_data data)
WRITE BINARY FILE START SIM CALLBACK.
- PLI_INT32 [write_binary_calltf](#) (PLI_BYTE8 *user_data)
Called by the simulator, each time it is requested. TODO.

2.5.1 Detailed Description

Functions to write raw binary files properly in verilog.

Author

Jay Convertino(johnathan.convertino.1@us.af.mil)

Date

2023-20-1

@LICENSE MIT Copyright 2023 Jay Convertino

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2.5.2 Function Documentation

2.5.2.1 write_binary_calltf()

```
PLI_INT32 write_binary_calltf (
    PLI_BYTE8 * user_data )
```

Called by the simulator, each time it is requested. TODO.

2.5.2.2 write_binary_start_sim_cb()

```
PLI_INT32 write_binary_start_sim_cb (
    p_cb_data data )
```

WRITE BINARY FILE START SIM CALLBACK.

2.5.2.3 write_thread()

```
void* write_thread (
    void * data )
```

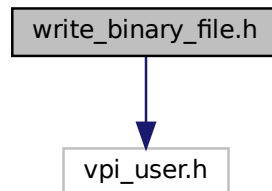
WRITE BINARY FILE THREAD TO EMPTY RINGBUFFER.

2.6 write_binary_file.h File Reference

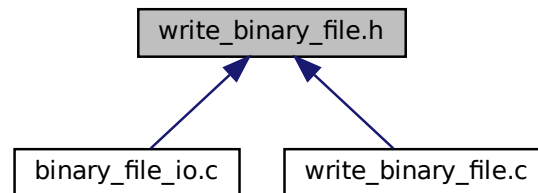
Functions to write raw binary files properly in verilog.

```
#include <vpi_user.h>
```

Include dependency graph for write_binary_file.h:



This graph shows which files directly or indirectly include this file:



Functions

- PLI_INT32 [write_binary_start_sim_cb](#) (p_cb_data data)
WRITE BINARY FILE START SIM CALLBACK.
- PLI_INT32 [write_binary_caltf](#) (PLI_BYTE8 *user_data)
Called by the simulator, each time it is requested. TODO.

2.6.1 Detailed Description

Functions to write raw binary files properly in verilog.

Author

Jay Convertino(johnathan.convertino.1@us.af.mil)

Date

2023-20-1

@LICENSE MIT Copyright 2023 Jay Convertino

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2.6.2 Function Documentation

2.6.2.1 write_binary_calltf()

```
PLI_INT32 write_binary_calltf (
    PLI_BYTE8 * user_data )
```

Called by the simulator, each time it is requested. TODO.

2.6.2.2 write_binary_start_sim_cb()

```
PLI_INT32 write_binary_start_sim_cb (
    p_cb_data data )
```

WRITE BINARY FILE START SIM CALLBACK.

Index

- arg2_handle
 - s_process_data, 1
- array_byte_size
 - s_process_data, 1
- binary_compiletf
 - binary_file_io.c, 4
- binary_end_compile_cb
 - binary_file_io.c, 4
 - binary_file_io.h, 8
- binary_end_sim_cb
 - binary_file_io.c, 4
 - binary_file_io.h, 8
- binary_file_io.c, 3
 - binary_compiletf, 4
 - binary_end_compile_cb, 4
 - binary_end_sim_cb, 4
 - binary_sizetf, 4
 - read_binary_reg_systf, 4
 - vlog_startup_routines, 5
 - write_binary_reg_systf, 5
- binary_file_io.h, 5
 - binary_end_compile_cb, 8
 - binary_end_sim_cb, 8
 - binary_sizetf, 8
 - BUFSIZE, 7
 - DATAHUNK, 7
 - READ_NAME, 7
 - WRITE_NAME, 7
- binary_sizetf
 - binary_file_io.c, 4
 - binary_file_io.h, 8
- BUFSIZE
 - binary_file_io.h, 7
- DATAHUNK
 - binary_file_io.h, 7
- error
 - s_process_data, 1
- num_ab_val_pairs
 - s_process_data, 2
- p_file
 - s_process_data, 2
- p_file_name
 - s_process_data, 2
- p_ringbuffer
 - s_process_data, 2
- read_binary_calltf
 - read_binary_file.c, 9
 - read_binary_file.h, 11
- read_binary_file.c, 8
 - read_binary_calltf, 9
 - read_binary_start_sim_cb, 9
 - read_thread, 10
- read_binary_file.h, 10
 - read_binary_calltf, 11
 - read_binary_start_sim_cb, 11
- read_binary_reg_systf
 - binary_file_io.c, 4
- read_binary_start_sim_cb
 - read_binary_file.c, 9
 - read_binary_file.h, 11
- READ_NAME
 - binary_file_io.h, 7
- read_thread
 - read_binary_file.c, 10
- s_process_data, 1
 - arg2_handle, 1
 - array_byte_size, 1
 - error, 1
 - num_ab_val_pairs, 2
 - p_file, 2
 - p_file_name, 2
 - p_ringbuffer, 2
 - systf_handle, 2
 - thread, 2
- systf_handle
 - s_process_data, 2
- thread
 - s_process_data, 2
- vlog_startup_routines
 - binary_file_io.c, 5
- write_binary_calltf
 - write_binary_file.c, 13
 - write_binary_file.h, 15
- write_binary_file.c, 12
 - write_binary_calltf, 13
 - write_binary_start_sim_cb, 13
 - write_thread, 13
- write_binary_file.h, 14
 - write_binary_calltf, 15
 - write_binary_start_sim_cb, 15
- write_binary_reg_systf

- binary_file_io.c, [5](#)
- write_binary_start_sim_cb
 - write_binary_file.c, [13](#)
 - write_binary_file.h, [15](#)
- WRITE_NAME
 - binary_file_io.h, [7](#)
- write_thread
 - write_binary_file.c, [13](#)