Linking Novas Files with Simulators and Enabling FSDB Dumping

Version O-2018.09-SP2, March 2019

SYNOPSYS®

Copyright Notice and Proprietary Information

Third-Party Software Notices

Verdi® Automated Debug Platform includes or is bundled with software licensed to Synopsys under free or open-source licenses. For additional information, see the third_party_notices.txt file in the *INSTALL_PATH/doc* directory of the Verdi software installation.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

www.synopsys.com

Contents

About This Book

Purpose

This book explains how to link the Novas object files for FSDB dumping with various standard simulators. The manual should be read from beginning to end, although you may skip any sections with which you are already familiar.

For detailed descriptions of commands for the Verdi platform, please refer to the appropriate chapter of the *Verdi and Siloti Command Reference Manual*.

Audience

The audience for this manual includes engineers who are familiar with languages and tools used in design and verification such as Verilog, VHDL, SystemVerilog, and typically install, set-up and run simulators.

This document assumes that you have a basic knowledge of the platform on which your version of the Verdi platform runs: Unix or Linux, and that you are knowledgeable in design and verification languages, simulation software, and digital logic design.

Book Organization

This *Linking Novas Files with Simulators and Enabling FSDB Dumping* is organized as follows:

- *About This Book* provides an introduction to this book and explains how to use it.
- *Overview* summarizes the basic information needed to generate FSDB files.
- *FSDB Dumping Commands* summarizes the common usage of FSDB dumping commands.
- *Linking with Cadence Simulators* summarizes the steps for linking the Novas object files with the Cadence IUS simulator for Verilog, VHDL and mixed designs.
- *Linking with ModelSim* Simulators summarizes the steps for linking the Novas object files with the ModelSim simulator.
- *Linking with Synopsys Simulators* summarizes the steps for linking the Novas object files with the Synopsys simulator.
- *Appendix A: Third Party Integration* provides examples for linking the object files for FSDB dumping with 3rd-party libraries.
- *Appendix B: Required VCS MX Options for Linking with FSDB Dumper* summarizes additional options to use with VCS MX when linking the object files for FSDB dumping.
- *Appendix C: Dumping to Multiple Files with FSDB Commands* provides details on using the FSDB dumping commands to dump to multiple FSDB files.
- *Appendix D: Switch Dumping with Simulator Restore* provides details on switching dumping between FSDB files while also using simulator restore commands.

Conventions Used in This Book

The following conventions are used in this book:

- *Italics font* is used for emphasizes, book titles, section names, design names, file path, and file names within paragraphs.
- **Bold** is used to emphasize text, highlight titles, menu options, and other Verdi terms.
- `Courier type` is used for program listings. It is also used for test messages that the Verdi platform displays on the screen.
- **NOTE** describes important information, warnings, or unique commands.
- **Menu->Option** identifies the path used to select a menu option.
- Left-click or click means left-click option on the indicated item.
- Middle-click means middle-click option on the indicated item.
- Right-click means right-click on the indicated item.
- Double-click means click twice consecutively with the left mouse button.
- Shift-left-click means press and hold the <Shift> key then click the left mouse button on the indicated item.
- Drag-left means press and hold the left mouse button, then move the pointer to the destination and release the button.
- Drag means press and hold the middle mouse button on the indicated item then move and drop the item to the other window.

Related Publications

- *Installation and System Administration Guide* - explains how to install the Verdi and Siloti systems.
- *Verdi and Siloti Quick Reference Guide* - provides a quick reference for using the Verdi and Siloti systems with typical debug scenarios.
- *Verdi and Siloti Command Reference Manual* - gives detailed information on the Verdi and Siloti command set.
- *Verdi User Guide and Tutorial* - provides detailed information on using the Verdi platform.
- *nAnalyzer User Guide and Tutorial* - provides detailed information on using the *nAnalyzer* Design Analysis module.
- *nECO User Guide and Tutorial* - provides detailed information on using the *nECO* Automated Netlist Modification module.
- *Siloti User Guide and Tutorial* - provides detailed information on using the Siloti system.
- *Release Notes* - for current information about the latest software version. Refer to the 'View release notes' link on the product downloads page.
- Language Documentation

  Hardware description (Verilog, VHDL, SystemVerilog, and so on) and verification language reference materials are not included in this manual. For language related documents, please refer to the appropriate language standards board (www.ieee.org, www.accellera.org) or vendor (www.synopsys.com, www.cadence.com) websites.

Overview

FSDB Format

The Verdi platform supports a file format called Fast Signal Database (FSDB) that has the following advantages over the standard VCD file format:

- An FSDB file is more compact than a standard VCD file. Typically, an FSDB file is about 5 to 50 times smaller than a VCD file.
- Using FSDB files, the Verdi platform displays waveform and back-annotated signal values faster.

Synopsys provides a set of object and support files in the Verdi package that can be linked with popular simulators to extend the simulator command set to support dumping FSDB files directly during simulation instead of VCD files. For existing VCD files, the `vfast` conversion utility is provided in the Verdi package and is used to convert VCD files into FSDB files. In addition, VCD files are converted to FSDB files automatically when a VCD file is opened using the **File -> Open Waveform File** command in *nTrace* or the **File -> Open** command in *nWave*. Another conversion utility, `fsdb2vcd`, is provided in the Verdi package that converts FSDB files back to VCD format.

If FSDB dumper support for a simulator version is over 18 months and there are already more than three major simulator versions for that vendor, that FSDB dumper is added into the legacy dumper.

For older simulator versions, please contact Synopsys support: https://solvnet.synopsys.com

FSDB Related Environment Variables

Refer to *Appendix A* in the *Verdi and Siloti Command Reference Manual* for details on FSDB dumping related environment variables.

NOTE: A *novas_dump.log* file gets created during simulation to record some essential simulation/dumping information, such as, the simulation options used, the environment variable settings, and the Novas object directories that were linked. This log file is needed to report a dumper related issue effectively.

Novas Object File Cross Reference Matrix

The following table summarizes which Novas object file directory should be used for which language and simulator version to enable FSDB dumping or interactive mode on specific platforms.

For Solaris, Linux, and IBM platforms:

Vendor	Product Name (Interface Type)	Supported Language	Supported Items 1. Simulator 2. Version 3. FSDB Dumping Platform
Synopsys	VCS (DKI/UCLI)	Verilog/ SystemVerilog/ SVA/ Verilog(SV)+/ VHDL Mixed	1. Synopsys VCS (VCSMX) 2. VCS MX 2015.09 or later 3. SOL2/SOL7/LINUX/ LINUX64/SUSE32/SUSE64
Cadence	IUS (PLI/FMI/CFC)	Verilog/ SystemVerilog/ SVA/VHDL/ Verilog(SV)+/ VHDL Mixed	1. Cadence NC-Sim 2. IUS 13.1 or later 3. SOL2/SOL7/LINUX/ LINUX64/IBM32/ IBM64/ SUSE32/SUSE64
Modeltech	MODELSIM (FLI/PLI)	Verilog/ SystemVerilog/ SVA/VHDL/ Verilog(SV)+/ VHDL Mixed	1. ModelTech ModelSim 2. ModelSim SE 10.2 or later 3. SOL2/SOL7/LINUX/ LINUX64/SUSE32/SUSE64

Subdirectory and Platform Mapping

The following table summarizes the object file sub-directories that should be used for different platforms.

Subdirectory ${PLATFORM}	Platform
SOL2 & SOLARIS2	SunOS 5.x (32 bit)
SOL7	SunOS 5.x (64 bit)
LINUX	Linux OS (32 bit)
LINUX64	Linux OS (64 bit)

Subdirectory ${PLATFORM}	Platform
IBM32	IBM-AIX 5.3 (32 bit)
IBM64	IBM-AIX 5.3 (64 bit)
SUSE32	SUSE OS (32 bit)
SUSE64	SUSE OS (64 bit)

FSDB Dumping Commands

FSDB Command Line, Environment Variable, and Dump Command Options

The unified FSDB dumper supports the following three methods for specifying options:

1. Specify an option on the simulator command line. Refer to the *FSDB Dumping Command Line Options* section for details on the command line options.
2. Specify an option using an environment variable. Refer to *Appendix A* in the *Verdi and Siloti Command Reference Manual* for details on the environment variables.
3. Specify an option in an FSDB dumping command. Refer to the FSDB Dumping Command Option table for details on using options with the dump commands.

If the same option is set using more than one method, the resolution is: Method 1 > Method 2 > Method 3.

For example, the FSDB dump file name can be set using one of the following methods:

a. Specify the file name on the simulator command line:

```
> simv +fsdbfile+high.fsdb
```

b. Specify the file name through an environment variable:

```
> setenv NOVAS_FSDB_FILE mid_prio.fsdb
```

c. Specify the file name in an FSDB dumping command:

```
$fsdbDumpvars("+fsdbfile+low_prio.fsdb");
```

That is, in this example, the resolved FSDB file name is "`high.fsdb`".

Mapping Command Line, Environment Variable, and Dumping Command Options

The following table maps the simulator command line options to their equivalent environment variables and FSDB dumping command options.

NOTE: An option which can be set on the simulator command line may not always have an equivalent mapping environment variable or FSDB dumping command option.

A *novas_dump.log* file gets created during simulation to record some essential simulation/dumping information, such as the simulation

options used, the environment variable settings, and the Novas object directories that are linked. This log file is needed to report a dumper-related issue effectively.

Table: Command Line, Environment Variable, and Dumping Command Option Mapping

Specify on Simulator Command Line	Specify Using Environment Variable	Specify With FSDB Dumping Command
+fsdb+delta	FSDB_DELTA	None
+fsdb+dump_limit=*size*	FSDB_DUMP_LIMIT	None
+fsdbfile+filename	NOVAS_FSDB_FILE	+fsdbfile+filename
+fsdbLogOff	None	None
+fsdbLog=*severity*	None	None
+fsdb+all[=*on*|*off*]	NOVAS_FSDB_ALL	+all
+fsdb+autoflush	None	None
+fsdb+dumpoff_glitch+time [+htime]	None	None
+fsdb+dumpon_glitch+time [+htime]	None	None
+fsdb+dumpoff_sequence *+time[+htime]* +fsdb+dumpoff_sequence *+time[time_unit]*	None	None
+fsdb+dumpon_sequence *+time[+htime]* +fsdb+dumpon_sequence *+time[time_unit]*	None	None
+fsdb+dump_log[=*on*|*off*]	FSDB_DUMP_LOG	None
+fsdb+dumpon+*time[+htime]*	None	None
+fsdb+dumpoff+*time[+htime]*	None	None

Specify on Simulator Command Line	Specify Using Environment Variable	Specify With FSDB Dumping Command
+fsdb+esconfig="*filename*" **NOTE:** This option and the **+fsdb+esoptions** option are exclusive. **NOTE:** When this option and the *ESCONFIG* or *FSDB_ESOPTIONS* environment variable are specified simultaneously, this option is ignored.	FSDB_ESCONFIG **NOTE:** When this environment variable and the *FSDB_ESOPTIONS* environment variables are specified simultaneously, the *FSDB_ESOPTIONS* environment variable is ignored.	None
+fsdb+esdb="*esdb_filename*"	FSDB_ESDB	None
+fsdb+esdb_es_only[=*on*|*off*]	FSDB_ESDB_ES_ONLY	None
+fsdb+esoptions="*ESAoption*" **NOTE:** This option and the **+fsdb+esconfig** option are exclusive. **NOTE:** When this option and the *FSDB_ESOPTIONS* or *ESCONFIG* environment variable are specified simultaneously, this option is ignored. **NOTE:** Refer to the table after the examples for all available ESA options.	FSDB_ESOPTIONS **NOTE:** When this environment variable and the *ESCONFIG* environment variables are specified simultaneously, the *FSDB_ESOPTIONS* environment variable is ignored.	None
+fsdb+force	FSDB_FORCE	None

Specify on Simulator Command Line	Specify Using Environment Variable	Specify With FSDB Dumping Command
+fsdb+functions **NOTE:** The **-debug_access+dmptf** option must be specified for VCS compile before setting the **+fsdb+functions** option. **NOTE:** The **-debug_access+all** option is also applicable in the place of **-debug_access+dumptf**. Starting from VCS 2017.12, the **+all** option includes **dumptf** option also.	FSDB_FUNCTIONS **NOTE:** The **-debug_access+dmptf** option must be specified for VCS compile before setting the *FSDB_FUNCTIONS* environment variable. **NOTE:** The **-debug_access+all** option is also applicable in the place of **-debug_access+dumptf**. Starting from VCS 2017.12, the **+all** option includes **dumptf** option also.	+functions
+fsdb+glitch=*num*	NOVAS_FSDB_ENV_MAX_GLITCH_NUM	None
+fsdb+ignore_vhdl_complex [=*on*|*off*]	NOVAS_FSDB_IGNORE_VHDL_COMPLEX	None
+fsdb+io_only	FSDB_IO_ONLY	+IO_Only
+fsdb+keep_cell_module=*module_name*	FSDB_KEEP_CELL_MODULE	None
+fsdb+max_var_elem=*num*	FSDB_MAX_VAR_ELEM	None
+fsdb+mda[=*on*|*off*]	NOVAS_FSDB_MDA	+mda
+fsdb+msv+i	FSDB_MSV_I	+msv+i
+fsdb+msv+i=	FSDB_MSV_IN *n*	+msv+i=n
+fsdb+msv+isub	FSDB_MSV_ISUB	+msv+isub
+fsdb+msv+v	FSDB_MSV_V	+msv+v
+fsdb+msv+v=	FSDB_MSV_VN *n*	+msv+v=n
+fsdb+msv+vaI	FSDB_MSV_VAI	+msv+vaI
+fsdb+msv+vaV	FSDB_MSV_VAV	+msv+vaV
+fsdb+msv+va	SDB_MSV_VA	+msv+va

Specify on Simulator Command Line	Specify Using Environment Variable	Specify With FSDB Dumping Command
+fsdb+packedmda[=*on*|*off*]	NOVAS_FSDB_PACKEDMDA	+packedmda
+fsdb+no_all_msg	None	None
+fsdb+no_error	None	None
+fsdb+no_msg_in_file+ filename	None	None
+fsdb+no_msg+ pattern1[+pattern2..]	None	None
+fsdb+no_parallel	FSDB_NO_PARALLEL	None
+fsdb+no_psl_scope	FSDB_NO_PSL_SCOPE	+no_psl_scope
+fsdb+no_warning	None	None
+fsdb+parameter[=*on*|*off*]	NOVAS_FSDB_ PARAMETER	+parameter
+fsdb+psl_prop	FSDB_DUMP_PSL_PROP	+psl_prop
+fsdb+reg_only	FSDB_REG_ONLY	+Reg_Only
+fsdb+region	FSDB_REGION	None
+fsdb+reserve	NOVAS_FSDB_RESERVE	None
+fsdb+reuse_filename	FSDB_REUSE_FILENAME	None
+fsdb+sequential	NOVAS_FSDB_ENV_DUMP_SEQ_NUM	None
+fsdb+skip_cell_instance=*mode*	NOVAS_FSDB_SKIP_CELL_ INSTANCE **Note**: The **-debug_region=cell** must be specified for VCS compiler before setting the *NOVAS_FSDB_SKIP_CELL_ INSTANCE* environment variable.	+skip_cell_ instance
+fsdb+strength[=*on*|*off*]	NOVAS_FSDB_STRENGTH	None
+fsdb+struct[=*on*|*off*]	NOVAS_FSDB_STRUCT	+struct
+fsdb+sva_index_info	FSDB_SVA_INDEX_INFO	None

Specify on Simulator Command Line	Specify Using Environment Variable	Specify With FSDB Dumping Command
+fsdb+sva_status	FSDB_SVA_STATUS	+sva_status
+fsdb+sva_success	FSDB_SVA_SUCCESS	None
+fsdb+time_precision=*time_unit*	FSDB_TIME_PRECISION	None
+fsdb+trace_process[=*on*|*off*]	NOVAS_FSDB_TRACE_PROCESS	+trace_process
+fsdb+trans_begin_callstack	NOVAS_FSDB_TRANS_BEGIN_CALLSTACK	None
+fsdb+trans_end_callstack	NOVAS_FSDB_TRANS_END_CALLSTACK	None
+fsdb+trans_error_callstack	FSDB_TRANS_ERROR_CALLSTACK	None
+fsdb+trans_hide_error	FSDB_TRANS_HIDE_ERROR	None
+fsdb+vhdl_package_lib=*+all|lib_name1:lib_name2:...*	FSDB_VHDL_PACKAGE_LIB	None
+fsdb+writer_mem_limit=*num*	NOVAS_FSDB_ENV_WRITER_MEM_LIMIT	None
+fsdb_trans_file+*filename*	FSDB_TRANS_FILE	None
+fsdb+virtual_file=filename	setenv FSDB_VIRTUAL_FILE	None
None	NOVAS_FSDB_DEFAULT_FILE_NAME	None

NOTE: **For Synopsys**:
The `+fsdb+sva_success` option can only be specified using the environment variable or the runtime option, as follows:

```
setenv FSDB_SVA_SUCCESS 1    //Supported
./simv <other options> +fsdb+sva_success   //
```
Supported

```
$fsdbDumpSVA
("+fsdb+sva_success","+fsdbfile+fsdb+sva_succes
```

```
s") //NOT supported
call{$fsdbDumpSVA("+fsdb+sva_success","+fsdbfil
e+fsdb+sva_success")}; //NOT supported
```

NOTE: **For Cadence**:
The `+fsdb+sva_success` option can only be specified using the environment variable or the runtime option, as follows:

```
setenv FSDB_SVA_SUCCESS 1       //Supported
ncsim <other options> fsdb+sva_success     //
Supported

fsdbDumpSVA +fsdb+sva_success //NOT supported
call $fsdbDumpSVA("+fsdb+sva_success"); //NOT
supported
```

Some options are related to one another in behavior. For example, the **NOVAS_FSDB_MDA** option can enable/disable MDA signal dumping and the **NOVAS_FSDB_ALL** option can enable/disable all supported signal type dumping. This makes **NOVAS_FSDB_ALL** a superset of **NOVAS_FSDB_MDA**. When such options are set using different methods at the same time, the effect of the setting is determined as follows:

1. Resolve each option's value using the simulator command line, environment variable, and then FSDB dumping command resolution priority.
2. Behave according to the union of all the option settings.

Example 1

1. Specify the environment variable to enable dumping MDA signals:
   ```
   setenv NOVAS_FSDB_MDA 1
   ```
2. Specify the option in an FSDB dumping command to enable dumping all supported signals:
   ```
   $fsdbDumpvars("+all");
   ```

It implies that the environment variable (1) should have priority over the setting in the FSDB dumping command (2). However, these two options are not the same, hence the resolution for the three methods cannot be applied.

The effect of the setting in this example is the union of the option setting of *NOVAS_FSDB_MDA(+fsdb+mda)* and *NOVAS_FSDB_ALL(+fsdb+all)* and that is “to enable dumping of all supported signals”.

Example 2

1. Specify the option on the simulator command line to enable dumping packed MDA signals:

   ```
   simv +fsdb+packedmda
   ```

2. Specify the environment variable to enable dumping MDA signals:

   ```
   setenv NOVAS_FSDB_MDA 1
   ```

The *NOVAS_FSDB_MDA* environment variable which is equal to the *+fsdb+mda* option is resolved to be *on*. The effect of the two related options *+fsdb+mda (=on)* and the *+fsdb+packedmda (=on)* are then unified as "to enable dumping of all MDA signals".

Example 3

Specify the options on the simulator command line to filter all dumper messages, warning messages, and error messages:

```
simv +fsdb+no_warning +fsdb+no_error +fsdb+no_all_msg
```

The effect of the three related options is the union of the three which is "to filter all dumper messages".

FSDB Dumping Command Line Options

The simulator command line options that can be used to configure FSDB dumping are summarized in the following table:

NOTE: These command line options are not available in pure VHDL designs.

Table: Simulation Command Line Options

Option	Description	Example
+fsdb+delta	Enables the following options, specifying the glitch value: **+fsdb+glitch=0** **+fsdb+sequential** **+fsdb+region** **NOTE:** If the **+fsdb+glitch** option is also enabled individually, the glitch value specified with the **+fsdb+glitch** option overwrites the glitch value specified with the **+fsdb+delta** option.	+fsdb+delta
+fsdb+dump_limit=*size*	Limits the size of the FSDB file. The default is *10MB*. **NOTE:** The minimum limit size is 10MB. If limit size is set to less than 10MB, the limit size is set to 10MB automatically.	+fsdb+dump_limit=15
+fsdbfile+*filename*	Specifies the FSDB file name. If it is not specified, the default FSDB file name is "*novas.fsdb*".	+fsdbfile+test1.fsdb
+fsdbLogOff	Avoids dumping logging messages.	+fsdbLogOff
+fsdbLog=*severity*	Dumps logging messages by level of severity.	+fsdbLog=3 +fsdbLog=-1 +fsdbLog=~5 ... +fsdbLog=7~ ... +fsdbLog=5~7 ... +fsdbLog=7~5 ...

Option	Description	Example
+fsdb+all[=*on*|*off*]	Enables dumping of all signals including memory, MDA, packed array, structure, union, power-related, and packed structure in all scopes specified in *$fsdbDumpvars* or the entire design if no scope is specified. The default is *off*.	+fsdb+all=on
+fsdb+power	Enables dumping of power-related signals. **NOTE**: Power-related signals include the power supply nets and power domain states. Power-related signals are dumped hierarchically under the **$power_root** folder. **NOTE**: If the *+fsdb+power* or *+fsdb+all=on* option is not specified, the power-related signals are not dumped. **NOTE:** This option is supported only with VCS simulator.	+fsdb+power
+fsdb+autoflush	Enables the automatic flush function when the simulation is stopped with the **$stop** dumping command or by using the Ctrl-C shortcut key. This option is useful when dumping the FSDB file in batch mode and to view the results on *nWave* immediately.	+fsdb+autoflush
+fsdb+dump_log [=*on*|*off*]	Enables/disables *novas_dump.log*. The default is *on*.	+fsdb+dump_log=on

Option	Description	Example
+fsdb+dumpoff_glitch +*time[+htime* or +fsdb+dumpoff_glitch +*time[time_unit*	Specifies the time {htime, time} to disable dumping of glitches. **NOTE:** The **+fsdb+glitch=num** option is required before this option works. **NOTE:** When the *time_unit* is not specified, the simulation time unit of Verilog is used as the default time unit. **NOTE:** If multiple dump off times are needed, repeat this option for each dump time.	+fsdb+dumpoff_ glitch +123+2 (Turn off dumping glitch commands at the specified time: {32'd<2>, 32'd<123>}) or +fsdb+dumpoff_ glitch +3000ps
+fsdb+dumpoff_sequence+*time[+htime]* or +fsdb+dumpoff_sequence+*time[time_unit]*	Turns *off* dumping sequence commands at the specified time {htime, time}. **NOTE:** When the *time_unit* is not specified, the simulation time unit of Verilog is used as the default time unit. **NOTE:** If multiple dump off times are needed, repeat this option for each dump time.	+fsdb+dumpoff_sequence +123+2 (Turn off dumping sequence commands at the specified time: {32'd<2>, 32'd<123>}) or +fsdb+dumpoff_sequence +3000ps
+fsdb+dumpoff+*time[+htime]* or +fsdb+dumpoff+*time[time_unit]*	Turns *off* dumping commands at the specified time {htime, time}. **NOTE:** When the *time_unit* is not specified, the simulation time unit of Verilog is used as the default time unit. **NOTE:** If multiple dump off times are needed, repeat this option for each dump time.	+fsdb+dumpoff+2500+2 (Turn off dumping sequence commands at the specified time: {32'd<2>, 32'd<2500>}) or +fsdb+dumpoff+2500ps
+fsdb+dumpon_glitch +*time[+htime]* or +fsdb+dumpon_glitch +*time[time_unit*]	Specifies the time {htime, time} to enable dumping of glitches. **NOTE:** The **+fsdb+glitch=num** option is required before this option works. **NOTE:** When the *time_unit* is not specified, the simulation time unit of Verilog is used as the default time unit. **NOTE:** If multiple dump off times are needed, repeat this option for each dump time.	+fsdb+dumpon_ glitch +123+2 (Turn on dumping glitch commands at the specified time: {32'd<2>, 32'd<123>}) or +fsdb+dumpon_ glitch +3000ps

Option	Description	Example
+fsdb+dumpon_sequence+*time[+htime]* or +fsdb+dumpon_sequence+*time[time_unit]*	Turns *on* dumping sequence commands at the specified time {htime, time}. **NOTE:** When the *time_unit* is not specified, the simulation time unit of Verilog is used as the default time unit. **NOTE:** If multiple dump on times are needed, repeat this option for each dump time.	+fsdb+dumpon_sequence +123+2 (Turn on dumping sequence commands at the specified time: {32'd<2>, 32'd<123>}) or +fsdb+dumpon_sequence +3000ps
+fsdb+dumpon+*time[+htime]* or +fsdb+dumpon+time[time_unit]	Turns *on* dumping commands at the specified time {htime, time}. **NOTE:** When the *time_unit* is not specified, the simulation time unit of Verilog is used as the default time unit. **NOTE:** If multiple dump on times are needed, repeat this option for each dump time.	+fsdb+dumpon+500+2 (Turn on dumping sequence commands at the specified time: {32'd<2>, 32'd<500>}) or +fsdb+dumpon+2500ps
+fsdb+esconfig="*filename*"	Specifies the ESA options in a text file. Options in the file can be separated by spaces or lines. Refer to the *ESA Options* section for available options.	+fsdb+esconfig="My_ESA_option.txt"
+fsdb+esdb="*esdb_filename*"	Specifies the ESDB file name. The FSDB dumper uses the essential signals in this file to determine which signals to dump. **NOTE:** *esdb_filename* is the file generated by the *esa* utility with the **-db** option.	+fsdb+esdb="es"
+fsdb+esdb_es_only [=*on*|*off*]	Enables Essential Signal dumping. The default is *on*.	+fsdb+esdb_es_only=on
+fsdb+esoptions="*ESA options*"	Specifies the ESA options. Refer to the *ESA Options* section for available options.	+fsdb+esoptions=" -xsignalfile my.list -xscope top"
+fsdb+force	Enables dumping the force status of the Verilog signal.	+fsdb+force

Option	Description	Example
+fsdb+functions	Set to enable dumping the signals in function and task. **NOTE:** The **-debug_access+dmptf** option must be specified for VCS compile before setting the **+fsdb+functions** option. **NOTE:** The **-debug_access+all** option is also applicable in the place of **-debug_access+dumptf**. Starting from VCS 2017.12, the **+all** option includes **dumptf** option also.	+fsdb+functions
+fsdb+glitch=*num*	Enables specifying the glitch number. Range: 0 ~ 254 Number: 0: All glitches are stored. 1: The last glitch is stored. 2: The first and the last glitches are stored. Refer to the *NOVAS_FSDB_ENV_MAX_GLITCH_NUM* environment variable for details.	+fsdb+glitch=0
+fsdb+ignore_vhdl_complex[=*on*|*off*]	Do not dump complex signal types for VHDL such as records, multi-dimensional arrays, and array of records.	+fsdb+ignore_vhdl_complex=on
+fsdb+io_only	Dumps only input/output port signals.	+fsdb+io_only

Option	Description	Example
+fsdb+keep_cell_module= *module_name*	Specifies the module that is not skipped when the **skip_cell_instance** option is turned *on*. The wildcard character (*) is supported. **NOTE:** This option can be specified multiple times in one simulation and multiple cell modules are kept. **NOTE:** When specifying this option multiple times, use the following format: *+fsdb+keep_cell_module=PD +fsdb+keep_cell_module=PD1 + fsdb+keep_cell_module=PD2* Using the following format reports an error: *+fsdb+keep_cell_module=PD+PD1+PD2*.	+fsdb+keep_cell_module=cell_b or +fsdb+keep_cell_module="cell_*"

+fsdb+max_var_elem= *num*	setenv FSDB_MAX_VAR_ELEM
	
	NOTE: When the *FSDB_MAX_VAR_ELEM* environment variable is specified, you can choose the upper boundary in order to get a better dumping performance.
	Example:
	• setenv FSDB_MAX_VAR_ELEM 100
	• simv +fsdb+max_var_elem=100
	Reg a[0:99]; // 100 elements, are dumped,
	Reg b[0:100]; // 101 elements, are not dumped.
		+fsdb+max_var_elem

+fsdb+mda[=*on*|*off*]	Enables/disables dumping all MDA signals. The default is *off*.	+fsdb+mda=off

Option	Description	Example
+fsdb+packedmda [=*on*|*off*]	Enables/disables dumping packed MDA signals. The default is *off*. **NOTE:** This option replaces the *+mda+packedOnly* option.	+fsdb+packedmda=on
+fsdb+msv+i	Dumps analog current signals. **NOTE:** Additional current voltage signals is dumped with this option. **NOTE:** It is required to turn *off* the shadow module optimization in the CustomSim initial file and compile the **-debug_access** option. **NOTE:** It is required to add the `iprobe_wavefore_current` in the CustomSim configure for dumping the current with this option.	+fsdb+msv+i
+fsdb+msv+i=[*n*|*all*]	Dumps the current of terminals. *n* is a positive integer. When `all` is specified, the currents of all terminals are dumped. **NOTE:** Additional analog current signals can be dumped with this option. **NOTE:** It is required to turn off the shadow module optimization in the CustomSim initial file and compile the **-debug_access** option. **NOTE:** It is required to add the `iprobe_wavefore_current` in the CustomSim configure for dumping the current with this option.	+fsdb+msv+i=1 +fsdb+msv+i=2 … or +fsdb+msv+i=all

Option	Description	Example
+fsdb+msv+v	Dumps analog voltage signals. **NOTE:** Additional analog voltage signals can be dumped with this option. **NOTE:** It is required to turn *off* the shadow module optimization in the CustomSim initial file and compile the **-debug_access** option.	+fsdb+msv+v
+fsdb+msv+v=[*n*|*all*]	Dumps the voltage of terminals. *n* is a positive integer. When all is specified, the voltages of all terminals are dumped. **NOTE:** Additional analog voltage signals can be dumped with this option. **NOTE:** It is required to turn *off* the shadow module optimization in the CustomSim initial file and compile the **-debug_access** option.	+fsdb+msv+v=1 +fsdb+msv+v=2 … or +fsdb+msv+v=all
+fsdb+msv+isub	Dumps the total current flows into the subcircuit port.	+fsdb+msv+isub
+fsdb+no_all_msg	Filters all FSDB dumper messages.	+fsdb+no_all_msg
+fsdb+no_error	Filters the error message.	+fsdb+no_error
+fsdb+no_msg_in_ file+*filename*	Filters the messages which are specified in a file (for example, *no_msg_list.txt*).	+fsdb+no_msg_in_file+ no_msg_list.txt
+fsdb+no_msg+ *pattern1*[+*pattern2*..]	Filters the messages which contain the specified strings (for example, "Analog").	+fsdb+no_msg+Analog
+fsdb+no_parallel	Disables parallel dumping.	+fsdb+no_parallel
+fsdb+no_psl_scope	Skips dumping the PSL scope named *cds_assertion_models* that is generated automatically in the simulator.	+fsdb+no_psl_scope
+fsdb+no_warning	Filters the warning message.	+fsdb+no_warning

Option	Description	Example
+fsdb+parameter [=*on*|*off*]	Enables/disables parameter dumping. The default is *off*.	+fsdb+parameter=on
+fsdb+psl_prop	Dumps the PSL property.	+fsdb+psl_prop
+fsdb+reg_only	Dumps only register type signals.	+fsdb+reg_only
+fsdb+region	Enables region mode dumping.	+fsdb+region
+fsdb+reserve	Do not overwrite the existing FSDB file. Upcoming FSDB files, which share the same name as the reserved one, must be re-named as `xxx_r0.fsdb`, instead.	+fsdb+reserve
+fsdb+reuse_filename	Reuse an existing FSDB filename if the file itself is removed while the simulation is running.	+fsdb+reuse_filename
+fsdb+sequential	Enables sequence dumping. Refer to the *FSDB_ENV_DUMP_SEQ_NUM* environment variable for details.	+fsdb+sequential
+fsdb+skip_cell_instance=*mode*	Enables/Disables dumping of cell instance where *mode* implies: 0: Disables functionality. 1: Skip dumping `celldefine and –v/-y library cells. 2: Skip dumping the content of `celldefine and –v/-y library cells but keep dumping ports. 3: Skip dumping `celldefine library cells and keep dumping –v/-y library cells. 4: Skip dumping `celldefine library cells but keep the ports of the `celldefine library cell. Keep dumping –v/-y library cells. Others: Show error message and ignore it.	+fsdb+skip_cell_instance=2

Option	Description	Example
+fsdb+strength [=*on*|*off*]	Enables/disables dumping strength. The default is *off*.	+fsdb+strength=on
+fsdb+struct[=*on*|*off*]	Enables/disables dumping struct signals. The default is *off*.	+fsdb+struct=on
+fsdb+sva_index_info	Dumps the assertion index information.	+fsdb+sva_index_info
+fsdb+sva_status	Dumps the assertion status.	+fsdb+sva_status
+fsdb+time_precision =*time_unit*	Manually set the time precision in the FSDB file.	+fsdb+timeprecision=1ns
+fsdb+trace_process [=*on*|*off*]	Enables/disables dumping VHDL variables under processes. The default is *off*. **NOTE:** For VCS users, the VCS debug level option needs to be set as "**-debug_all**" for dumping process variables.	+fsdb+trace_process=on
+fsdb+trans_error_ callstack	Dumps the full call stack for each transaction error message from the Novas object files for FSDB dumping.	+fsdb+trans_error_ callstack
+fsdb+trans_hide_ error	Hides all transaction error messages from the Novas object files for FSDB dumping.	+fsdb+trans_hide_ error
+fsdb+vhdl_package_lib=+*all* or +fsdb+vhdl_package_lib=*lib_name1:lib_name2*:...	Dumps the packages under all libraries or the specified libraries. **NOTE:** If the library is not specified, no packages are dumped.	+fsdb+vhdl_package_lib=+all or +fsdb+vhdl_package_lib=work:std:ieee

Option	Description	Example
+fsdb+writer_mem_limit=*num*	Specifies the limit for the FSDB writer to flush value changes in memory. The default value is *64* when the FSDB file has less than 5M signals. The default value is *128* when the FSDB file has 5M - 10M signals. The default value is *256* when the FSDB file has more than 10M signals. Valid range is from 4 to 2048. The units are in MB.	+fsdb+writer_mem_limit=64
+fsdb_trans_file+*filename*	Specifies a transaction FSDB file. **NOTE:** When this option and the **+fsdbfile+filename** option are specified simultaneously, this option is ignored.	+fsdb_trans_file+test1.fsdb
+fsdb+virtual_file	Creates a virtual file that contains all FSDB created during simulation.	+fsdb+virtual_file=all.vf

ESA Options

The ESA options used for `+fsdb+esoptions`=*"ESAoption"* are summarized in the following table. These options are optional.

Table: ESA Options Supported in Simulator Runtime Phase

Option Syntax	Description
-dumpclk *none(n)|all(a)*	Specify the clock and reset signal dumping option. The default is *all*.
-exclude_floating *0|1*	Exclude floating net if set to 1. The default is *0*.
-exclude_inst_file *filename*	Exclude signals inside the instances specified in the file.
-exclude_nd_module *0|1*	If outputs of library cells cannot be computed, automatically dump the outputs and exclude the internals of the cell. The default is *0*.

Option Syntax	Description
-hier_replace *source_scope=target_scope*	Replace the current hierarchy name of the scope with the specified hierarchy name.
-IO *instance*	Dump I/O ports of instances.
-IO_level *N*	Specify the level for **-IO**. This is optional. The default is *N=1*.
-suppress_mfile *filename*	Suppress Essential Signal dumping for all modules listed in the specified file.
-xscope *instance*	Dump all signals for the specified instance.

-xscopefile *filename*	Dump all signals for the instances specified in the file. Multiple scopes with different levels can be specified. Each pair of the level-scope is separated by a line. If the level of scopes is not specified, its value is the same as the `-xscope_level` value (the default is *1*). The format of *xscopefile.f* is:
	`2 top.a`
	`0 top.b`
	`top.c`
	
	Example:
	`simv +vcs+lic+wait +fsdb+esdb=es +fsdb+esoptions="-xscope tb_CPUsystem.i_CPUsystem -xscope_level 1 -xscopefile xscopefile.f"`
	
	In the *xscopefile.f*:
	2 tb_CPUsystem.i_CPUsystem.i _CPU

-xscope_level *N*	Specify the level for **-xscope**. The default is *N=1*.
-xsignalfile *filename*	Dump the signals from the specified legacy hierarchical Essential Signal list (text format). **NOTE:** To dump the signals from the legacy plain Essential Signal list, use *$fsdbDumpvarsByFile*.

FSDB Dumping Commands Used with Verilog

The Novas object files for FSDB dumping provide the following FSDB dumping commands for Verilog:

$fsdbAutoSwitchDumpfile
$fsdbDumpfile
$fsdbDumpflush
$fsdbDumpon, $fsdbDumpoff
$fsdbDumpvars
$fsdbDumpvarsByFile
$fsdbDumpFinish
$fsdbDumpMDA
$fsdbDumpMDAByFile
$fsdbDumpPSL
$fsdbDumpSVA
$fsdbLog
$fsdbSuppress
$fsdbSwitchDumpfile

Supported Simulators for FSDB Dumping Commands

This table illustrates the supported simulator version for each FSDB dumping command.

Verilog Task Name	IUS 13.1 and later	VCS 2015.09 and later	ModelSim 10.2 and later
$fsdbAutoSwitchDumpfile	x	x	x
$fsdbDumpfile	x	x	x
$fsdbDumpflush	x	x	x
$fsdbDumpoff	x	x	x
$fsdbDumpon	x	x	x
$fsdbDumpvars	x	x	x
$fsdbDumpvarsByFile	x	x	x
$fsdbDumpFinish	x	x	x
$fsdbDumpMDA	x	x	x

$fsdbDumpSVA	x	x	x
$fsdbLog	x	x	x
$fsdbSuppress	x	x	x
$fsdbSwitchDumpfile	x	x	x

General Dumping Commands

$fsdbDumpfile

Description

It specifies the FSDB file name created by the Novas object files for FSDB dumping. If it is not specified, then the default FSDB file name is *"novas.fsdb"*. This command is valid only before executing *$fsdbDumpvars* and is ignored if specified after *$fsdbDumpvars*.

To restrict the largest dump file size to an user-defined size limitation, you must specify the limit size in megabyte format. This function impacts the FSDB file that is specified in *"FSDB_Name"*. The FSDB dumper uses the sliding window scheme to keep the latest signal values in the FSDB file. It discards the old values if the file size exceeds the user-specified limitation.

For *$fsdbDumpvars*, *$fsdbDumpMDA*, *$fsdbDumpSVA* and *$fsdbDumpvarsByFile*, you can use the option *"+fsdbfile+filename.fsdb"* to specify the target FSDB file for dumping. It is equivalent to using *$fsdbDumpfile* and then *$fsdbDumpvars*.

NOTE: To leverage the memory consumption, the maximum number of opened FSDB files for dumping in a single simulation run should be no more than 32 files.

Syntax

When specified in the design:

```
$fsdbDumpfile("FSDB_Name" | FSDB_Name_var [ ,Limit_Size |
,Limit_Size_var ]);
```

When specified on the simulator command line:

```
Synopsys:
fsdbDumpfile "FSDB_Name"
fsdbDumpfile "FSDB_Name" Limit_Size
```

Cadence:
```
call fsdbDumpfile "FSDB_Name"
call fsdbDumpfile "FSDB_Name" Limit_Size
```

ModelSim:
```
fsdbDumpfile "FSDB_Name"
fsdbDumpfile "FSDB_Name" Limit_Size
```

Arguments

`FSDB_Name`

Specify the name of the FSDB file generated by the Novas object files for FSDB dumping.

`FSDB_Name_var`

Specify the FSDB file name in a variable.

NOTE: Valid only if the command is specified inside the design.

`Limit_Size`

Specify the maximum FSDB file size in megabyte.

`Limit_Size_var`

Specify the maximum FSDB file size in a variable.

NOTE: Valid only if the command is specified inside the design.

NOTE: The minimum file size of the FSDB file is 10M. If you set the size to less than 10M, it is set to 10M instead.

Examples

NOTE: The following example uses the syntax for calling the FSDB dumping command in the design. Refer to the syntax section for the correct format for the simulator command line.

```
$fsdbDumpfile("novas.fsdb", 32);
/* Specify dump file as novas.fsdb and limit to 32 megabytes */
$fsdbDumpvars(0, design_top);
```

Variable Example

If specified in the design, the design variable could be passed as command argument.

```
reg [32*8-1:0] option_reg = "+fsdbfile+novas.fsdb";
reg [15:0] limit_size_reg = 1000;
string newDumpfile_string = "novas_new.fsdb";
reg [1023:0] newDumpfile_reg = "novas_new.fsdb";
```

```
$fsdbDumpfile("novas.fsdb");
$fsdbDumpfile(newDumpfile_string, 1000);
$fsdbDumpfile(newDumpfile_reg, limit_size_reg);
```

$fsdbDumpFinish

Description

This command closes all FSDB files in the current simulation and stops dumping of signals. Although all FSDB files are closed automatically at the end of simulation, this dumping command can be invoked to explicitly close the FSDB files during the simulation. A new FSDB file can be created after the old FSDB file is closed. New dumping data cannot be appended to a closed FSDB file due to the current limitation of the FSDB file architecture. If a closed FSDB file name is opened again during the same simulation run, then the new FSDB file is renamed automatically.

Syntax

When specified in the design:

```
$fsdbDumpFinish;
```

When specified on the simulator command line:

Synopsys:
```
fsdbDumpFinish
```

Cadence:
```
call fsdbDumpFinish
```

ModelSim:
```
fsdbDumpFinish
```

$fsdbDumpflush

Description

This FSDB dumping command can be used in the HDL code and also on the command line for interactive simulation control. This command forces the signal values to be flushed to the FSDB file while the simulation is running. Therefore, the current simulation results can be checked whenever necessary instead of waiting for the simulation to complete or the internal buffer to be filled.

NOTE: The Novas object files for FSDB dumping automatically flushes the value change data to the FSDB file on the disk based on following conditions:
- The simulation stop is due to a *$stop* statement in the design.
- The simulation stops at a user specified breakpoint.
- The user uses *Ctrl-C* to break the simulation.

Syntax

When specified in the design:

```
$fsdbDumpflush;
```

When specified on the simulator command line:

Synopsys:
```
fsdbDumpflush
```

Cadence:
```
call fsdbDumpflush
```

ModelSim:
```
fsdbDumpflush
```

$fsdbDumpon, $fsdbDumpoff

Description

These FSDB dumping commands are used to turn on or turn off dumping. *$fsdbDumpon* and *$fsdbDumpoff* have the highest priority and can override all other FSDB dumping commands such as *$fsdbDumpvars*, *$fsdbDumpMDA*, *$fsdbDumpSVA*, or *$fsdbDumpvarsByFile*.

In the *nWave* window, blue solid blocks are displayed during the period when dumping is turned off by the *$fsdbDumpoff* command and no signals are dumped. All signals are assigned with the **NV** (No Value) data value in the value pane of

the *nWave* window and for **Active Annotation** in other windows. In addition, the full scale ruler marks this period with red box.

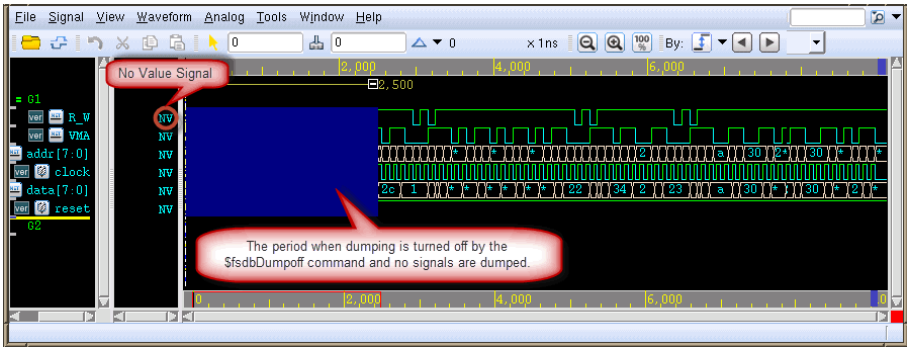


Figure: Dump-Off Period in the nWave Window - No Value Signal

The signals that have not started dumping values when the FSDB file specified by the *$fsdbDumpfile* command are opened. They are assigned with the **NF** (Not Found) data value in the value pane of the *nWave* window.

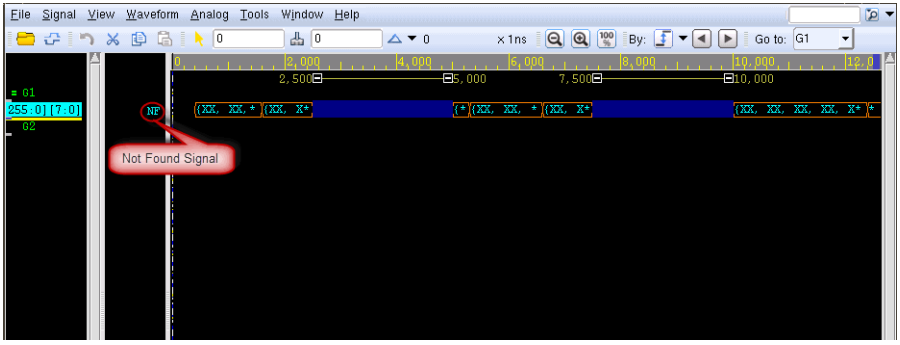


Figure: Dump-Off Period in the nWave Window - Not Found Signal

NOTE: *$fsdbDumpon* and *$fsdbDumpoff* are not restricted to only *$fsdbDumpvars*.
If multiple FSDB files are open for dumping at one simulation run, *$fsdbDumpon* and *$fsdbDumpoff* may only affect a specific FSDB file by specifying the specific file name.

NOTE: *$fsdbDumpon* and *$fsdbDumpoff* only support integers as the argument for time.

Syntax

When specified in the design:

```
$fsdbDumpon([ "option"| option_var| ]);
$fsdbDumpoff([ "option"| option_var]);
```

When specified on the simulator command line:

Synopsys:

```
fsdbDumpon [option]
fsdbDumpoff [option]
```

Cadence:

```
call fsdbDumpon [option]
call fsdbDumpoff [option]
```

ModelSim:

```
fsdbDumpon [option]
fsdbDumpoff [option]
```

Arguments

`option`

+fsdbfile+*filename*	For the *$fsdbDumpon* command, specify the target FSDB file name for "Dump On". If it is not specified, dumping for all FSDB files is turned on.
	For the *$fsdbDumpoff* command, specify the target FSDB file name for "Dump off". If it is not specified, dumping for all FSDB files is turned off.
"+glitch"	For the *$fsdbDumpon* command, specify the time to enable dumping of glitches. **NOTE:** The **+fsdb+glitch=num** option is required before this option works. The **+fsdb+glitch=num** option specifies the glitch number ranging from 0 to 254.
	For the *$fsdbDumpoff* command, specify the time to disable dumping of glitches. **NOTE:** The **+fsdb+glitch=num** option is required before this option works. The **+fsdb+glitch=num** option specifies the glitch number ranging from 0 to 254.

`option_var`

Specify the FSDB file name option in a variable.

NOTE: It is valid only if the command is specified inside the design.

Examples

NOTE: The following examples use the syntax for calling the FSDB dumping command in the design. Refer to the syntax section for the correct format on the simulator command line.

```
initial
  begin
  #50000
  $fsdbDumpvars;
  #50000
  $fsdbDumpoff;
  #50000
  $fsdbDumpon;
  #50000
  $finish;
end
```

The value changes for all variables in this design example are dumped from time 50000 to time 100000 and from time 150000 to time 200000.

```
initial  begin
  $fsdbDumpoff("+glitch")
  #10
  $fsdbDumpvars;
  #10
  $fsdbDumpon("+glitch");
  #10
  $fsdbDumpoff("+glitch");
  $finish;
end
```

Any glitches in this design example are enabled for dumping between the time instances 20 and 30.

Variable Example

If specified in the design, design variable can be passed as an argument.

```
reg [32*8-1:0] option_reg = "+fsdbfile+novas.fsdb";
string option_string = "+fsdbfile+novas.fsdb";

$fsdbDumpon();
$fsdbDumpon("+fsdbfile+novas.fsdb");
$fsdbDumpon(option_reg);
$fsdbDumpon(option_string);

reg [32*8-1:0] option_reg = "+fsdbfile+novas.fsdb";
string option_string = "+fsdbfile+novas.fsdb";

$fsdbDumpoff();
$fsdbDumpoff("+fsdbfile+novas.fsdb");
$fsdbDumpoff(option_reg);
```

```
$fsdbDumpoff(option_string);
```

$fsdbDumpvars

Description

This command dumps the change in signal value to the FSDB file.

NOTE:

1. For VCS users, to include memory, MDA, packed array and structure information in the generated FSDB file, the **-debug_access** option or alternative **+vpi**, **+memcbk** and **+vcsd** options must be included when VCS is invoked to compile the design.
2. The generated FSDB can be converted to VCD which includes the MDA and structure data.

Syntax

When specified in the design:

```
$fsdbDumpvars([ depth, | "level=",depth_var, ]
        [instance | "instance=",instance_var]
        [ ,"option" | ,"option=",option_var ]*);

$fsdbDumpvars;
$fsdbDumpvars([depth] [, instance ]);
$fsdbDumpvars("level=", depth_var [, instance ]);
$fsdbDumpvars( ["level=", depth_var] [,"option"]*);
$fsdbDumpvars([depth] [,"option=", option_var]*);
```

NOTE:

1. Dumping design variables only supports dumping commands that have been specified inside the source code of the design. In the case of interactive mode, use the task in the simulator command line format.
2. After a dumping command has been specified inside the design, it is compatible with both dumping variable and simulator interactive mode formats.

When specified on the simulator command line:

Synopsys:

```
fsdbDumpvars
fsdbDumpvars  depth instance [option]*
fsdbDumpvars depth [instance]
fsdbDumpvars [option]*
```

Cadence:

```
call fsdbDumpvars
call fsdbDumpvars  depth instance [option]*
call fsdbDumpvars  [option]*
```

```
Modelsim:
fsdbDumpvars
fsdbDumpvars  depth instance [option]*
```

Arguments

`depth`

Specifies the number of levels of hierarchy to dump for the subsequent scopes. If the subsequent arguments are signals, only those signals are dumped.

NOTE: When specified in the design, if the depth argument is specified as a variable type, the first argument must be specified as a string literal *"level="* and the second argument as level argument. The default level value is 0. Level values have the following meanings:
0: all signals in all scopes.
1: all signals in current scope.
2: all signals in the current scope and all scopes one level below.
n: all signals in the current scope and all scopes n-1 levels below.

`"level="`

Keyword to identify the next argument is a number.

NOTE: It is valid only if the command is specified inside the design.

`depth_var`

Specifies the number of levels of hierarchy to dump for the subsequent scopes in a variable.

NOTE: It is valid only if the command is specified inside the design.

`instance`

This definition gives the module scope or signals in full hierarchy format that specify the objects to dump.

NOTE: If this FSDB dump command is included in a VHDL design, double quotes are required for all signals under a VHDL scope. If this FSDB dump command is included in a Verilog design, double quotes are not required.

`"instance="`

Keyword to identify the content of the next argument is a module scope or signal.

NOTE: It is valid only if the command is specified inside the design.

`instance_var`

This definition provides the module scope or signal represented by instance_var which specifies the object to dump.

NOTE: It is valid only if the command is specified inside the design.

`option`

Specify one or more of the following options.

"+IE_Only"	Only IE signals are dumped. This option dumps both the analog and digital signal of all types of IEs such as, d2a, a2d, n2e, e2n, r2e, and e2r.
"+IO_Only"	Only IO port signals are dumped.
"+Reg_Only"	Only reg type signals are dumped.
"+mda"	Dumps all memory and MDA signals in all scopes specified in *$fsdbDumpvars* or the entire design if no scope is specified. This option is covered by the "*+all*" option and is effective for SystemVerilog's MDAs only. For VCS users, the VCS option "*+memcbk*" may be needed while compiling the design. **NOTE:** This option does not apply to VHDL arrays because they are dumped by default without this option. **Example:** *$fsdbDumpvars("+mda");*
"+msv+v=N"	Dumps signal voltage for the specified terminal.
"+msv+v=all"	Dumps signal voltage for all terminals.
"+msv+i=N"	Dumps signal current for the specified terminal.
"+msv+i=all"	Dumps signal current for all terminals.
"+packedmda"	Dumps the packed signals in the design. Refer to the *Supported MDA Format Table* for a summary of which packed or unpacked one-dimensional/two-dimensional signals are dumped with or without the *+packedmda* and *+mda* options. This option is covered by the "*+mda*" option. **Example:** *$fsdbDumpvars("+packedmda");* **NOTE:** This option replaces the *+mda+packedOnly* option.
+packedmda+struct	Dumps the struct or the struct in packed MDA. **Example:** *$fsdbDumpvars("+packedmda+struct ");*

"+struct"	Dumps all structs in all scopes specified in *fsdbDumpvars* or in the entire design if no scope is specified. This option is covered by the "+*all*" option and is effective for SystemVerilog struct syntax only. **Example:** *$fsdbDumpvars("+struct");*
"+skip_cell_instance =*mode*"	Enables/Disables dumping of cell instance where *mode* means: 0: disable functionality. 1: Skip all cell info. 2: Dump all ports of cell instance. Others: Show error message and ignore it. **Example:** *$fsdbDumpvars("+skip_cell_instance");*
"+all"	Dumps all signals including memory, MDA, packed array, structure, union, power-related, and packed structure in all scopes specified in *$fsdbDumpvars* or the entire design if no scope is specified. For VCS users, the VCS option *"+memcbk"* may be needed when compiling the design. **NOTE**: Power-related signals include power supply nets and power domain states. Power-related signals are dumped hierarchically under the **$power_root** folder. **NOTE**: If the "+*all*" or "+*power*" options are not specified, the power-related signals are not dumped.
"+parameter"	Dumps parameters. **Example:** *$fsdbDumpvars("+parameter");* **NOTE**: If the parameter type is "scalar", "real", or "string", the name is displayed without range information.
"+power"	Dumps power-related signals. **Example:** *$fsdbDumpvars("+power");* **NOTE**: Power-related signals include the power supply nets and power domain states. Power-related signals are dumped hierarchically under the **$power_root** folder. **NOTE**: If the "+*power*" or "+*all*" option is not specified, the power-related signals are not dumped.
"+trace_process"	Dumps VHDL processes. Example: *$fsdbDumpvars("+trace_process");*
"+functions"	Enables dumping of functions in the design. **Example:** *$fsdbDumpvars("+functions");*
"+fsdbfile+*filename*"	Specifies the FSDB file name. If not specified, the default FSDB file name is "*novas.fsdb*". **NOTE:** An environment variable can be used as part of the filename string argument. Refer to the *Use Environment Variable(s) in the String Argument* section for an example. When an environment variable is defined and then embedded into the option, the FSDB dumper replaces the environment variable with the given definition.

+msv+va	Dumps the Verilog-A/Verilog-AMS parameter probe. The signals dumped by the command line are integer, real, genvar, and parameter. **NOTE:** An error message appears and the Verilog-A/ Verilog-AMS probes are not dumped if the `-iprobe_waveform_va` command is not specified in the CustomSim configuration file.
+msv+vaV	Dumps the Verilog-A/Verilog-AMS voltage probe. The signals dumped by the command line are module port voltage, terminal voltage, and branch voltage. **NOTE:** An error message appears and the Verilog-A/ Verilog-AMS probes are not dumped if the `-iprobe_waveform_va -branch_voltage` command is not specified in the CustomSim configuration file.
+msv+vaI	Dumps the Verilog-A/Verilog-AMS current probe. Module port current, terminal current, and branch current. **NOTE:** An error message appears and the Verilog-A/ Verilog-AMS probes are not dumped if the Parameter probe `-iprobe_waveform_va -branch_current` and `iprobe_waveform_current` command is not specified in the CustomSim configuration file.

Supported MDA Format Table

Options	Unpacked 1D e.g. T a[ ];	Unpacked 2D e.g. T a[ ][ ];	Packed 1D e.g. T [ ]a;	Packed 2D e.g. T [ ][ ]a;
None			Yes	
"+packedmda"			Yes	Yes
"+mda"	Yes	Yes	Yes	Yes

`"option="`

Keyword to identify the next argument is a string.

NOTE: Valid only if the command is specified inside the design.

`option_var`

Specify the option in a variable.

NOTE: Valid only if the command is specified inside the design.

Examples

NOTE: The following example uses the syntax for calling the FSDB dumping command in the design. Refer to the syntax section for the correct format for the simulator command line.

```
$fsdbDumpvars;
```

Dumps all signals in the module instances below the top module.

```
$fsdbDumpvars(0, system);
```

Dumps all signals in the module instances below *system*.

```
$fsdbDumpvars(0, system, "+fsdbfile+novas.fsdb");
```

Dumps all signals in the module instances below system to the specified FSDB file named "novas.fsdb"

```
$fsdbDumpvars(1, top.dut1.u1.clk);
```

Dumps the clk signal under *top.dut1.u1* when top and u1 are Verilog scopes and dut1 is a VHDL scope. This command is included in a Verilog design.

```
$fsdbDumpvars("level=", level_reg, system);
```

Dumps all signals in the system instance and module instances below it up to the depth that *level_reg* variable indicates.

```
$fsdbDumpvars("+fsdbfile+my.fsdb")
```

Dumps all signals of all scopes to the specified FSDB file, “my.fsdb”.

Variable Example

When specified in the design, a runtime variable can be passed as an argument.

```
reg [255:0] depth_reg = 10;
reg [32*8-1:0] option_reg = "+fsdbfile+novas.fsdb";
string option_string = "+fsdbfile+novas.fsdb";
string depth_string = "10";// invalid, number in string format
reg [32*8-1:0] depth_str_reg = "10"; // invalid.
reg [1023:0] instance_reg = "top";
string instance_string = "top";

$fsdbDumpvars(1, top, "+fsdbfile+novas.fsdb");
$fsdbDumpvars("level=", depth_reg);
$fsdbDumpvars("option=", option_reg);
$fsdbDumpvars("option=", option_string);
$fsdbDumpvars("level=", depth_reg, "option=", option_reg);
$fsdbDumpvars("level=", depth_reg, top, "option=",
option_string);
$fsdbDumpvars("instance=", instance_reg);
$fsdbDumpvars(1, "instance=", instance_string);
```

The dumping command of *$fsdbDumpvars(0, design_top, "+fsdbfile+novas.fsdb");* is equal to the combination of the two dumping commands: *$fsdbDumpfile("novas.fsdb"); $fsdbDumpvars(0, design_top);*

NOTE: Refer to the *$fsdbSuppress* FSDB dumping command for details on suppressing individual signals or scopes from the design.

Use Environment Variable(s) in the String Argument

Instead of modifying the file name path with each simulation run, an environment variable can be added to the argument and then the value of the environment variable can be changed for each simulation run. For example, assume a variable path location is desired for the +fsdbfile+filename string argument in the *$fsdbDumpvars* dumping command and the following is specified:

- On the Unix command line:

setenv MY_FSDB_FILE_PATH /home/test

- In the testbench for simulation:

$fsdbDumpvars("+fsdbfile+${MY_FSDB_FILE_PATH}/test.fsdb");

- Multiple environment variables in the testbench for simulation:

setenv MY_FSDB_FILE_PATH /home/test

setenv MY_FSDB_FILE test

$fsdbDumpvars("+fsdbfile+${MY_FSDB_FILE_PATH}/${MY_FSDB_FILE}.fsdb");

Then, during simulation, the FSDB dumper replaces ${MY_FSDB_FILE_PATH} with "/home/test", and the dumping command becomes $fsdbDumpvars("+fsdbfile+/home/test/test.fsdb").

$fsdbDumpvarsByFile

Description

Dump several scopes and signals defined in a text file to a designated FSDB file. *$fsdbDumpvarsByFile* can be invoked more than once to dump several scopes/signals defined in different text files to specific FSDB files. There are two schemes to specify designated FSDB files:

1. No FSDB file name specified in *$fsdbDumpvarsByFile* (only one argument needed). If no FSDB file is specified, the default FSDB file name is, *novas.fsdb*, or the previously specified file name is the default file name.

2. Specify the FSDB file name in *$fsdbDumpvarsByFile* (two arguments needed). The argument "+fsdbfile+filename.fsdb" can be used to specify the designated FSDB file.

NOTE: " #" can be used to comment within the text file used by *$fsdbDumpvarsByFile*. The items after " #" are ignored.

Syntax

When specified in the design:

```
$fsdbDumpvarsByFile ("textFileName" | textfile_var [,"option"
|,option_var ]);

$fsdbDumpvarsByFile("textFileName");
$fsdbDumpvarsByFile("textFileName", "option");
```

When specified on the simulator command line:

Synopsys:
```
fsdbDumpvarsByFile "textFileName" [option]
```

Cadence:
```
call fsdbDumpvarsByFile "textFileName" [option]
```

ModelSim:
```
fsdbDumpvarsByFile "textFileName" [option]
```

Arguments

`"textFileName"`

Specify the name of the text file containing the scopes to dump (in plain text format).

If a packed element is specified in the `textFileName` of `$fsdbDumpvarsByFile`, then:

- It dumps the whole packed element in the case Verdi 2017.12.
- It dumps only the exact element specified with env `FSDB_PARTIAL_PACKED`.

`textfile_var`

Specify the text file name in a variable.

NOTE: Valid only if the command is specified inside the design.

`option`

+fsdbfile+*filename*	Specify the FSDB file name. If not specified, the default FSDB file name is "*novas.fsdb*".

`option_var`

Specify the FSDB file name option in a variable.

NOTE: Valid only if the command is specified inside the design.

File Format of the Text File

```
# [pragma]
# comment
[option_in_file]* [depth] instance_name
```

Similar to *$fsdbDumpvars*, the options in the following table are for the text file only. They can only be specified in the [`option_in_file`] location of the "textfile" only. For example:

```
+all 1 system
+mda +struct 3 system.i_cpu
```

"+IO_Only"	Only IO port signals are dumped.
"+functions"	Enables dumping of functions in the design.
"+Reg_Only"	Only reg type signals are dumped.
"+mda"	Dumps all memory and MDA signals in all scopes specified in $fsdbDumpvars or the entire design if no scope is specified. This option is covered by the *"+all"* option and is effective for SystemVerilog's MDAs only. For VCS users, the VCS option *"+memcbk"* may be needed when compiling the design.
"+msv+v=N"	Dumps signal voltage for the specified terminal.
"+msv+v=all"	Dumps signal voltage for all terminals.
"+msv+v"	Dumps analog voltage signals.
"+msv+i=N"	Dumps signal current for the specified terminal.
"+msv+i=all"	Dumps signal current for all terminals.

"+msv+i"	Dumps analog current signals.
"+packedmda"	Only dumps the packed signals in the design (without array and memory types). The *Supported MDA Format Table* summarizes if the packed or unpacked one-dimensional/two-dimensional signals are dumped with or without the *+packedmda* and *+mda* options. **Example:** *$fsdbDumpvars("+packedmda");* **NOTE:** This option replaces the *+mda+packedOnly* option.
+packedmda+struct	Dumps the struct or the struct in packed MDA. **Example:** *$fsdbDumpvars("+packedmda+struct ");*
"+struct"	Dumps all structs in all scopes specified in *fsdbDumpvars* or in the entire design if no scope is specified. This option is covered by the "*+all*" option and is effective for SystemVerilog struct syntax only. **Example:** *$fsdbDumpvars("+struct");*
"+skip_cell_instance=*mode*"	Enable/Disable dumping of cell instance where *mode* means: 0: disable functionality. 1: Skip all cell info. 2: Dump all ports of cell instance. Others: Show error message and ignore it.
"+all"	Dumps all signals including the memory, MDA, packed array, structure, power related signals, union and packed structure signals in all scopes specified in $fsdbDumpvars or the entire design if no scope is specified. For VCS users, the VCS option *"+memcbk"* may be needed when compiling the design.
"+parameter"	Dumps parameters. **NOTE:** If the parameter type is “scalar”, “real”, or “string”, the name is displayed without range information.
"+trace_process"	Dumps VHDL process.
"+power"	Dumps power-related signals. **Example:** *$fsdbDumpvars("+power");* **NOTE**: Power-related signals include the power supply nets and power domain states. Power-related signals are dumped hierarchically under the **$power_root** folder. **NOTE**: If the "*+power*" or "*+all*" options are not specified, the power-related signals are not dumped.

The following pragma can optionally be specified as the first line in the text file:

#__NOESD__	Do not perform Essential Signal Dumping (ESD) with the dumping command that uses the file with this pragma included.

Examples

NOTE: The following example uses the syntax for calling the FSDB dumping command in the design. Refer to the syntax section for the correct format for the simulator command line.

```
$fsdbDumpvarsByFile("./case1.lst","+fsdbfile+case1.fsdb");
```

where case1.lst contains:

```
[case1.lst]
+Reg_Only 0 top.type_dut.regInst
+parameter 0 top.type_dut.parameterInst
+functions 5 top.type_dut.FunAndTaskInst
+skip_cell_instance=0 0 top.type_dut.tri0Inst
+skip_cell_instance=1 +mda 0 top.type_dut.tri1Inst
+skip_cell_instance=2 +packedmda 0 top.type_dut.triInst
+mda 3 top.type_dut.enumInst
+packedmda 3 top.type_dut.logicInst
+struct top.type_dut.pstructInst
```

Variable Example

Specified in the design, the runtime variable can be passed as an argument.

```
...
reg [32*8-1:0] option_reg = "+fsdbfile+novas.fsdb";
reg [1023:0] suppressFile_reg = "suppress.list"; // reg string
string dumpListFile_string = "dump.list";//string
     ...

$fsdbDumpvarsByFile("dump.list", "+fsdbfile+novas.fsdb");
$fsdbDumpvarsByFile(dumpListFile_reg);
$fsdbDumpvarsByFile(dumpListFile_string, option_reg);
```

$fsdbLog

Description

Records a message to the specified message stream. Each message stream is organized under a scope named "msg_root" in the FSDB file. For example:

```
msg_root
              |--- status_stream
```

```
|--- compare_result_stream
```

NOTE: To use *$fsdbLog*, two simulator compile options must be specified for VCS. One is **-debug_pp** and the other is +**cli+3**. For example:
```
vcs -debug_pp +cli+3 -sverilog test.sv -P $tab
$pli
```

NOTE: *$fsdbLog* does not support the complex expression as the argument. If users want to pass a complex expression to *$fsdbLog*, a variable can be used to save the result and pass the expression to *$fsdbLog*. For example:
```
$fsdbLog((stats_ctl ? "CTL" : "NCTL"),
"GMX",,"inb_stats", "val");
```
can be changed to
```
string label = (stats_ctl ? "CTL" : "NCTL");
$fsdbLog(label, "GMX",,"inb_stats", "val");
```

Syntax

```
$fsdbLog(["label"], ["message"], [severity], ["stream_name"]
[["format"]*, variables | "string"]* );
```

NOTE: *$fsdbLog* can only be specified inside the design.

NOTE: The comma (",") for each argument cannot be omitted even if the argument is not specified. For example:
```
$fsdbLog( , , , , var1);
```

Arguments

`label`

Specify the label for the message in the string. This argument is optional and can be empty. If a label is not specified, a default label "*Unnamed Label*" is given. If the label needs to be formatted, *$psprintf* can be passed as the argument.

`message`

Specify the message in the string. This argument is optional and can be empty. If this argument is empty, an empty message is written to the message attribute in the record of the message stream.

If the message needs to be formatted, *$psprintf* can be passed as the argument. Format tokens, such as %d, %b, %h, in the message are not currently supported. For example:

```
$fsdbLog("label", "var1=%d", 1,  "stream1", var1);
```
Message is logged as "var1=%d" instead of "var1=3".

```
$fsdbLog("label", $psprintf("var1=%d", var1), 1, "stream1");
```
Message is logged as "var1=3".

```
$fsdbLog("label", "", "stream1", var1);
```
Message is logged as "".

`severity`

Specify the severity as a positive integer, negative integer, or empty. Severity is recorded as an attribute in the message record. When the severity is empty, no severity attribute is dumped. This argument is optional and can be empty.

`stream_name`

Specify the stream name to be created if it does not already exist. If the stream name needs to be formatted, *$psprintf* can be passed as the argument. This argument is optional and can be empty. Below are two examples.

Example 1

```
$fsdbLog called inside p.classAInst.funA.
```
The stream name is assigned as "p_classAInst".

```
$fsdbLog called inside
p.classAInst.funB.classBInst.funC.funD.
```
The stream name is assigned as "p_classAInst_classBInst".

Example 2

```
"\test.a.b.c* " or "test\.a\.b\.c\* "
```

Legal stream name rules are composed of 0-9, a-z, and A-Z. The '\' should be added in the prefix when the name contains a special character. Alternatively, the '\' can be added before each special character.

NOTE: Any of the first four arguments can be empty. However, the commas between the empty arguments should be placed properly. For example:
`$fsdbLog( , , , , var1);`
or
`$fsdbLog(, , , );`

`format`

Specify whether the logged information with the specified radix is shown on *nWave* or *nTA*. If the radix is not specified, the logged information is displayed as the default format of the variable. The format is shown below.

Supported format: format ::= %h, %x , %b, %d, %o, %A, ["str_name" | str_var]
%h ::=hexadecimal format

%x ::=hexadecimal format
%b ::=binary format
%d ::=decimal format
%o ::=octal format
%A ::=change the attribute name

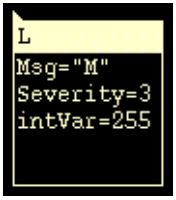
NOTE: When the format "%A" is used, the next argument must be a string value.

Example 1

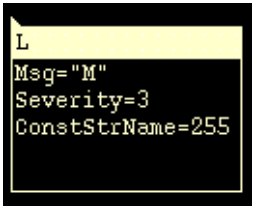
```
program p;
  class Foo1;
    function new (int i);
      ...
      $fsdbLog("label","message",1,...);
      ...
  endclass
  ...
  $fsdbLog("label","message",1,"stream_name","%h",info);
//specified radix with 'hexadecimal' format
  ...
  $fsdbLog("label","message",2,"stream_name",info2);
  ...
endprogram
```

Example 2

```
int intVar = 255;
$fsdbLog("L", "M", 3, "S", intVar);
```

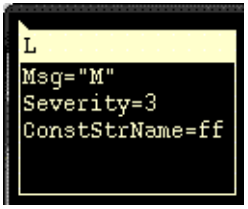



```
$fsdbLog("L", "M", 3, "S", "%A", "ConstStrName", intVar);
```

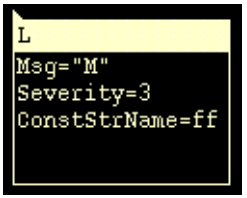



```
//specified radix with related format
```

```
$fsdbLog("L", "M", 3, "S", "%A", "ConstStrName", "%h",
intVar);
```




```
$fsdbLog("L", "M", 3, "S", "%h", "%A", "ConstStrName",
intVar);
```



`variable`

Specify the variable to log. An object with the vcsd_handle type can be passed here; however, the variable type is limited to string, longint, time integer, int, shortint, byte, logic, bit, reg, and real. In addition, an array element can be passed if the type of the element is one of the supported variable types listed above. For example, int i[0:1][0:3]; i[1][3] is allowed.

`string`

Specify a constant string to log. An attribute name for the constant string is automatically assigned.

Multiple string arguments are allowed. However, the total number of string and variable arguments is limited to less than 1024.

Each string argument is added as an attribute to the message record:

```
attribute_name = value
```

where the attribute_name is automatically assigned. The first one is named "Const_Msg0", and the second one is named "Const_Msg1". The value is the string itself. For example:

```
virtual function cpu_word_queue_t pack();
        pack.push_back(operation); // 8 bit field
        if (mode != NoOperand) begin
            pack.push_back(operand); // 8 bit field
        end
$fsdbLog(
//Label
```

```
    $psprintf("About to pack %s",operation.name()),
    //Message & severity
    "cpu_word_queue_t",5,
    //stream name
    $psprintf("%m"),
    //string #1
    $psprintf("op=%x, operand=%x ",operation,operand),
    //string #2
    $psprintf("Mode=%s,Size=%d",mode.name(),pack.size()
    ));
    ...
    endfunction: pack
```

Examples

Example 1

```
function void func(ref logic [2:0] gg);
if(gg==3'b101) begin
gg = 3'b111;
  $fsdbLog("REF","changed","3","ref","gg",$bits(gg));
  $display("[%0t]REF-changed-gg=%b",$time,gg);
end else
$fsdbLog("REF","unchanged",,"ref",gg);
$display("[%0t]REF-unchanged-gg=%b",$time,gg);
endfunction
```

Example 2

```
class c_type2;
    rand int i;
endclass

class c_type1;
  typedef enum reg [1:0] {ONE=1,TWO,THREE} number;
  number num;
  rand reg [1:0][1:0] r_mda1;
  reg [1:0] mda1;
  c_type2 c2 = new;
  function void disp;
    $fsdbLog("L1",
              $psprintf("Time:[%0t] num=%d(%d)",$time,num,THREE),
              1,
              "",
              num,
              THREE,
              "This is RAW string1",
              $psprintf("%d",c2.randomize()),
              $psprintf("%d",r_mda1),
              mda1);
    endfunction
endclass
```

Limit FSDB Size

$fsdbAutoSwitchDumpfile

Description

Automatically switch to a new dump file when the working FSDB file reaches the specified size limitation. The unit associated with *$fsdbAutoSwitchDumpfile*'s *File_Size* parameter is a megabyte. After the dumping is finished, a virtual FSDB file is automatically created and list all of the generated FSDB files with the correct sequence. Only the virtual FSDB file, rather than all of the FSDB files, needs to be loaded to view the simulation results.

Syntax

When specified in the design:

```
$fsdbAutoSwitchDumpfile(File_Size | File_Size_var, "FSDB_Name" |
FSDB_Name_var, Number_of_Files | Number_of_Files_var
[ ,"log_filename" | ,log_filename_var ], ["+fsdb+no_overwrite"]);
```

When specified on the simulator command line:

Synopsys:
```
fsdbAutoSwitchDumpfile File_Size FSDB_name Number_of_Files
[log_filename]
```

Cadence:
```
call fsdbAutoSwitchDumpfile File_Size FSDB_name Number_of_Files
[log_filename]
```

ModelSim:
```
fsdbAutoSwitchDumpfile File_Size FSDB_name Number_of_Files
[log_filename]
```

Arguments

`File_Size`

The FSDB file size limit.

NOTE: The minimum file size of the FSDB file is 10M. If the size is set to less than 10M, it is set to 10M automatically.

`File_Size_var`

Specify the FSDB file size in a variable.

NOTE: Valid only if the command is specified inside the design.

`FSDB_Name`

Specify the FSDB file generated by the Novas object files for FSDB dumping.

`FSDB_Name_var`

Specify the FSDB file name in a variable.

NOTE: Valid only if the command is specified inside the design.

`Number_of_Files`

The maximum number of FSDB files to generate. When the *+no_overwrite* option is also used, this option specifies the maximum number of FSDB files to be dumped.

NOTE: If the number is set as 0, the dumper creates new FSDB files without file number limitation.

`Number_of_Files_var`

Specify the maximum number of FSDB files to generate in a variable. When the *+no_overwrite* option is also used, the variable specifies the maximum number of FSDB files to be dumped.

NOTE: Valid only if the command is specified inside the design.

`Log_filename`

This argument is optional. Specify the file name for the log file.

`log_filename_var`

Specify the log file name in a variable.

NOTE: Valid only if the command is specified inside the design.

`+fsdb+no_overwrite`

Stop dumping to the FSDB file when the file number restriction is reached.

NOTE: Specify the file number using the option *Number_of_Files* or *Number_of_Files_var* before using this option.

Examples

NOTE: The following examples use the syntax for calling the FSDB dumping command in the design. Refer to the syntax section for the correct format for the simulator command line.

```
$fsdbAutoSwitchDumpfile(10, "test.fsdb", 20);
$fsdbDumpvars(0, system);
```

The first FSDB file is named as *test_000.fsdb*. The second FSDB file *test_001.fsdb* is created when the file size of *test_000.fsdb* reaches near 10MB. The third FSDB file *test_002.fsdb* is created when the file size of *test_001.fsdb* reaches near 10MB, and so on. The first FSDB file *test_000.fsdb* is overwritten when the file size of *test_019.fsdb* reaches near 10MB. The default *test.log* file lists the time range of each FSDB file.

```
$fsdbAutoSwitchDumpfile(10, "test.fsdb", 20, "my.log");
$fsdbDumpvars(0, system);
```

The number and the maximum file size of the FSDB file are as the same as the previous example, but the log file is set to *my.log*.

Variable Example

If specified in the design, the design variable can be passed as a command argument.

```
reg [255:0] File_Size_reg = 50;
reg [255:0] File_Size_reg1 = "50";
/* Not support number in string format */
string FSDB_Name _string = "FSDB_Name.fsdb";
reg [32*8-1:0] option_reg = "+fsdbfile+novas.fsdb";
reg [15:0] Number_of_Files_reg = 1000;
string log_filename_string = "fsdb.log";
reg [1023:0] FSDB_Name_reg = "FSDB_Name.fsdb";

$fsdbAutoSwitchDumpfile(50, "FSDB_Name", 1000, "fsdb.log");
$fsdbAutoSwitchDumpfile(File_Size_reg, FSDB_Name_string, 1000);
$fsdbAutoSwitchDumpfile(50, FSDB_Name_reg, Number_of_Files_reg,
log_filename_string);
```

$fsdbSuppress

Description

Specify the scopes that are not dumped by the *$fsdbDumpvars*, *$fsdbDumpvarsByFile*, or *$fsdbDumpMDA* commands. *$fsdbSuppress* supports suppressing by file name or the target scopes passed as arguments. The supported target scope could be modules, instances, scopes, or signals in full hierarchical format. Each format has to be called in different *$fsdbSuppress* calls.

NOTE: *$fsdbSuppress* must be specified before *$fsdbDumpvars*, *$fsdbDumpvarsByFile*, *$fsdbDumpMDA*; otherwise, *$fsdbSuppress* does not work.

NOTE: The setting of *$fsdbSuppress* is reset after *$fsdbDumpFinish* is executed.

Syntax

When specified in the design:

```
$fsdbSuppress("suppress_file" | "file=",file_var | [instance]*, |
"instance=" [, instance_var ]* | "module_file=", "filename" |
"module_base="[, "module_name"]* | "signal_prefix=");

$fsdbSuppress("suppress_file");
$fsdbSuppress(instance[,instance]*);
$fsdbSuppress("instance=",instance_var[,instance_var]*);
$fsdbSuppress("file=", file_var);
$fsdbSuppress("module_file=","filename");
$fsdbSuppress("module_base=","module_name"[,"module_name"]*);
```

```
$fsdbSuppress("signal_prefix=");
```

NOTE: If the variable field options are used, the variable must be assigned before being called.

When specified on simulator command line:

Synopsys:
```
fsdbSuppress "suppress_file"
fsdbSuppress instance [instance]*
```

Cadence:
```
call fsdbSuppress "suppress_file"
call fsdbSuppress instance [instance]*
```

ModelSim:
```
fsdbSuppress "suppress_file"
fsdbSuppress instance [instance]*
```

Arguments

`suppress_file`

Specify the file containing the names of the scopes that are not dumped.

NOTE: For the suppress file format, each line of the file is the full name of a scope that you do not want to be dumped.

`instance`

Specify one or more scopes or signals that are not dumped.

`file=`

Keyword to identify the next argument is a string.

NOTE: Valid only if the command is specified inside the design.

`file_var`

Specify the suppress file name in a variable.

NOTE: Valid only if the command is specified inside the design.

`"instance="`

Keyword to identify the content of the next argument is a module scope or signal.

NOTE: Valid only if the command is specified inside the design.

`instance_var`

Gives the module scope or signal represented by instance_var that specifies the object to dump.

NOTE: The option, *"instance="* is needed only if providing a full hierarchical scope (or signal) in variable format. For those requirements of scope in explicitly plain string format, pass them as an argument, no *"instance="* option needed.

`"module_file="`

Keyword to identify the content of the next argument is a string.

`"filename"`

Specify the file containing the names of the modules that are not dumped.

`"module_base="`

Keyword to identify the content of the next argument(s) is a module.

`"module_name"`

Specify the module from which the instantiated scopes are not dumped.

NOTE: All supported suppression methods should be used separately for each *$fsdbSuppress* called; multiple suppression types cannot be combined in one *$fsdbSuppress* command.

`"signal_prefix="`

Specify the prefix pattern. Signals that match this pattern are not dumped.

Examples

NOTE: The following examples use the syntax for calling the FSDB dumping command in the design. Refer to the syntax section for the correct format for the simulator command line.

```
$fsdbSuppress(dump_top,system.i_pram);
$fsdbDumpvars(0,dump_top);

$fsdbSuppress("./suppress.lst");
```

where suppress.list contains:

```
[suppress.lst]
system.R_W
system.i_cpu.C0
system.i_cpu.C1
system.i_cpu.C5
system.i_cpu.C6
...
```

NOTE: The scopes to suppress, passed as task parameters or listed in the suppression list, support both instance and module format.

Variable Example

Specified in the design, the runtime variable can be passed as an argument.

```
...
reg [1023:0] suppressFile_reg = "suppress.list";
reg [1023:0] suppressFile_reg = "suppress.list"; // reg string
string suppressFile_string = "suppress.list";
string instance_string = "scope_a";
reg [1023:0] instance_reg = "scope_b";
...

$fsdbSuppress("suppress.list");
$fsdbSuppress(top.u1.u2);
$fsdbSuppress(top.u1.u2, top.x1.x2);
$fsdbSuppress("file=", suppressFile_reg);
$fsdbSuppress("file=", suppressFile_string);
//full hierarchy variable.
$fsdbsuppress("instance=", instance_string, instance_reg);
$fsdbSuppress("file=", suppressFile_string);     // list file
variable
$fsdbSuppress("module_file=", "suppress_module.list");
$fsdbSuppress("module_base=", "module_1", "module_2",
"module_3");
$fsdbSuppress("signal_prefix=", "_zy", "_zz", "zz_", "zy_");
```

$fsdbSwitchDumpfile

Description

Support multiple FSDB files during a simulation run by specifying the currently active FSDB file name. When this command is executed, the previously opened FSDB file is closed and the new FSDB file is created to continue saving the values change data and act as the current active FSDB file.

Since the FSDB dumper may support more then one open file for saving data simultaneously, the source file that is supposed to be switched can be specified by the option *"+fsdbfile+sourcefilename"*, otherwise the source file would be the currently active FSDB file.

Syntax

When specified in the design:

```
$fsdbSwitchDumpfile("NewFileName" | NewFileName_var
[ ,"+fsdbfile+src_file"| ,src_file_var ]*);
```

When specified on the simulator command line:

Synopsys:
```
fsdbSwitchDumpfile "NewFileName"
fsdbSwitchDumpfile "NewFileName" +fsdbfile+src_file
```

Cadence:
```
call fsdbSwitchDumpfile "NewFileName"
call fsdbSwitchDumpfile "NewFileName" "+fsdbfile+src_file"
```

ModelSim:
```
fsdbSwitchDumpfile "NewFileName"
fsdbSwitchDumpfile "NewFileName" "+fsdbfile+src_file"
```

Arguments

`NewFileName`

The new name of the FSDB file generated by the Novas object files for FSDB dumping.

`NewFileName_var`

Specify the new FSDB file name in a variable.

NOTE: Valid only if the command is specified inside the design.

`+fsdbfile+src_file`

Specify the source FSDB file to be switched (include the *.fsdb* file extension in the name). This is optional; if not specified, the FSDB dumper switches dump finish on all active FSDB files.

`src_file_var`

Specify the FSDB source file option in a variable.

NOTE: Valid only if the command is specified inside the design.

Examples

NOTE: The following example uses the syntax for calling the FSDB dumping command in the design. Refer to the syntax section for the correct format for the simulator command line.

Example 1

```
$fsdbDumpfile("test0.fsdb");
$fsdbDumpvars(0, system);
$fsdbDumpvars(3, system, "+fsdbfile+f1.fsdb);
#1000
$fsdbSwitchDumpfile("test1.fsdb", "+fsdbfile+test0.fsdb");
#1000
```

```
$fsdbSwitchDumpfile("test2.fsdb", "+fsdbfile+f1.fsdb");
```

Create *test0.fsdb* to dump the design initially. Create *f1.fsdb* to dump a different depth of design. Switch *test0.fsdb* to *test1.fsdb* at simulation time 1000. Then switch *f1.fsdb* to *test2.fsdb* at simulation time 2000.

Example 2

If specified in the design, the design variable can be passed as a command argument.

```
reg [32*8-1:0] option_reg = "+fsdbfile+novas.fsdb";
string newDumpfile_string = "novas_new.fsdb";
reg [1023:0] newDumpfile_reg = "novas_new.fsdb";

$fsdbSwitchDumpfile("novas_new.fsdb");
/* switch current file to novas_new.fsdb */

$fsdbSwitchDumpfile(newDumpfile_reg, "+fsdbfile+novas.fsdb");
/* switch novas.fsdb file to be novas_new.fsdb */

$fsdbSwitchDumpfile(newDumpfile_string, option_reg);
/* switch novas.fsdb to novas_new.fsdb */
```

Memory/MDA Dumping Command

$fsdbDumpMDA

Description

Dumps the value changes of MDA (multidimensional array) signals to the FSDB file. All memories and MDAs within the specified instance are dumped. In addition, it also supports partial range dumping of the data related to the value change of a specific MDA instance into the FSDB file.

NOTE:

1. For VCS users, to include memory, MDA, packed array, and structure information in the generated FSDB file, the VCS option **-debug_pp** is required when VCS is invoked to compile the design.
2. The generated FSDB can be converted to VCD which includes the MDA and structure data.

Syntax

When specified in the design:

```
$fsdbDumpMDA([ depth, | "level=",depth_var, ]
[instance | "instance=", instance_var ]
```

```
[ ,"option" | ,"option=",option_var ]*);

or

$fsdbDumpMDA( mda_instance [, 1st_dim_begin_index [, 1st_dim_size
                           [, 2nd_dim_begin_index [, 2nd_dim_size
                           ...]]]]);

or

$fsdbDumpMDA ("mda_instance=", mda_instance_var,
               [, "begin_index=", 1st_dim_begin_index
               [, "dim_size=", 1st_dim_size
               [, "begin_index=", 2nd_dim_begin_index
               [, "dim_size=", 2nd_dim_size...]]]]);
```

When specified on the simulator command line:

```
Synopsys:
fsdbDumpMDA
fsdbDumpMDA [depth] [instance] ["option"]*
or
fsdbDumpMDA mda_instance [1st_dim_begin_index [1st_dim_size
                         [2nd_dim_begin_index [2nd_dim_size
                         ...]]]]

Cadence:
call fsdbDumpMDA [instance] ["option"]*
call fsdbDumpMDA [depth] [instance]
or
call fsdbDumpMDA mda_instance [1st_dim_begin_index [1st_dim_size
                              [2nd_dim_begin_index [2nd_dim_size
                              ...]]]]

Modelsim:
fsdbDumpMDA
fsdbDumpMDA [depth] [instance] ["option"]*
or
fsdbDumpMDA mda_instance [1st_dim_begin_index [1st_dim_size
                         [2nd_dim_begin_index [2nd_dim_size
                         ...]]]]
```

Arguments

`depth`

Specify how many levels of hierarchy to dump for the subsequent scopes.

`"level="`

Keyword to identify the next argument is a number.

NOTE: Valid only if the command is specified inside the design.

`depth_var`

Specify how many levels of hierarchy to dump for the subsequent scopes in a variable.

NOTE: Valid only if the command is specified inside the design.

`instance`

Give the module scope or signal that contains the MDA objects to dump.

`"instance="`

Keyword to identify the content of the next argument is a module scope or signal.

`instance_var`

Give the module scope or signal represented by instance_var that specifies the object to dump.

`option`

+fsdbfile+*filename*	Specify the FSDB file name. If not specified, the default FSDB file name is "*novas.fsdb*".
"+skip_cell_instance=*mode*"	Enable/Disable dumping of cell instance where *mode* means: 0: Disable functionality. 1 or 2: Skip all cell information. Others: Show error message and ignore it.

Supported MDA Format Table

Unpacked 1D e.g. T a[ ];	Unpacked 2D e.g. T a[ ][ ];	Packed 1D e.g. T [ ]a;	Packed 2D e.g. T [ ][ ]a;
Yes	Yes	No	Yes

`"option="`

Keyword to identify the next argument is a string.

NOTE: Valid only if the command is specified inside the design.

`option_var`

Specify the option in a variable.

NOTE: Valid only if the command is specified inside the design.

`mda_instance`

The target MDA instance to select ranges from.

`Nth_dim_begin_index`

The beginning index number of the Nth dimension.

`Nth_dim_size`

The size of the Nth dimension to dump.

NOTE: Partially selecting MDA's dimensions is valid for SystemVerilog MDA statements only. It is not valid if used by the VHDL procedure call, *fsdbDumpMDA*.

Examples

NOTE: The following example uses the syntax for calling the FSDB dumping command in the design. Refer to the syntax section for the correct format for the simulator command line.

Assume the following memory and MDA signal are defined:

```
reg [7:0] screen [79:0][0:24]
```

Dump value changes of MDA (multidimensional array) signals to the FSDB file:

Dumping Command	Dumping Result
`$fsdbDumpMDA(screen);`	Dump all cells of MDA screen.
`$fsdbDumpMDA(screen, 10);`	Dump screen[10:10][0:24][7:0]
`$fsdbDumpMDA(screen, 10, 1);`	Dump screen[10:10][0:24][7:0]
`$fsdbDumpMDA(screen, 10, 1, 12);`	Dump screen[10:10][12:12][7:0]
`$fsdbDumpMDA(screen, 10, 1, 12, 2);`	Dump screen[10:10][12:13][7:0]
`$fsdbDumpMDA(screen, 10, 1, 12, 2, 7);`	The packed dimension is not supported.
`$fsdbDumpMDA(screen, 10, 1, 12, 2, 7, 3);`	The packed dimension is not supported.

`$fsdbDumpMDA(system);`

Dump all memories and MDAs under the system scope and its child scopes.

```
$fsdbDumpMDA(1, system.i_pram);
```

Dump all memories and MDAs under the system.pram scope.

```
$fsdbDumpMDA(0, top, "+skip_cell_instance=1",
"+fsdbfile+dump.fsdb");
```

Dump all memories and MDAs under top scope and skip cell instance.

Variable Example

If specified in the design, the design variable can be passed as an argument.

```
reg [255:0] depth_reg = 10;
reg [32*8-1:0] option_reg = "+fsdbfile+novas.fsdb";
string option_string = "+fsdbfile+novas.fsdb";
string depth_string = "10";// invalid, number in string format
reg [32*8-1:0] depth_str_reg = "10"; // invalid.
reg [1023:0] instance_reg = "top";

$fsdbDumpMDA(1, top, "+fsdbfile+novas.fsdb");
$fsdbDumpMDA("level=", depth_reg, "option=", option_string);
$fsdbDumpMDA("level=", depth_reg, top, "option=", option_reg);
$fsdbDumpMDA("instance=", instance_reg);
```

$fsdbDumpMDAByFile

Description

Dumps the value changes of MDA (multidimensional array) signals in a text file to a designated FSDB file. *$fsdbDumpMDAByFile* can be invoked more than once to dump several scopes and signals defined in different text files to specific FSDB files. There are two schemes to specify designated FSDB files:

1. No FSDB file name specified in *$fsdbDumpMDAByFile* (only one argument needed). If no FSDB file is specified, the default file, *novas.fsdb*, or the previously specified one is taken as the default.
2. Specify the FSDB file name in *$fsdbDumpMDAByFile* (two arguments needed). The argument `+fsdbfile+filename.fsdb` can be used to specify the designated FSDB file.

Syntax

When specified in the design:

```
$fsdbDumpMDAByFile ("textFileName" | textfile_var [,"option"
|,option_var ]);

$fsdbDumpMDAByFile("textFileName");
$fsdbDumpMDAByFile("textFileName", "option");
```

When specified on the simulator command line:

Synopsys:
```
fsdbDumpMDAByFile "textFileName" [option]
```

Cadence:
```
call fsdbDumpMDAByFile "textFileName" [option]
```

ModelSim:
```
fsdbDumpMDAByFile "textFileName" [option]
```

Arguments

`"textFileName"`

Specify the name of the text file containing the scopes to dump (in plain text format).

`textfile_var`

Specify the text file name in a variable.

NOTE: Valid only if the command is specified inside the design.

`option`

+fsdbfile+*filename*	Specify the FSDB file name. If not specified, the default FSDB file name is "*novas.fsdb*".

Supported MDA Format Table

Unpacked 1D e.g. T a[ ];	Unpacked 2D e.g. T a[ ][ ];	Packed 1D e.g. T [ ]a;	Packed 2D e.g. T [ ][ ]a;
Yes	Yes	No	Yes

`option_var`

Specify the FSDB file name option in a variable.

NOTE: Valid only if the command is specified inside the design.

File Format of the Text File

```
# comment
# Specify the scope instance:
[option_in_file] [depth] instance

# Specify the MDA variable:
[option_in_file] mda_variable [1st_dim_begin_idx
[1st_dim_size [2st_dim_begin_idx [2st_dim_size]]]]
```

Similar to *$fsdbDumpMDA*, the option in the following table is for the text file only. It can only be specified in the [`option_in_file`] location of the "textfile". For example:

```
+skip_cell_instance=0 1 system
system.i_cpu
```

"+skip_cell_instance=*mode*"	Enable/Disable dumping of cell instance where *mode* means: 0: disable functionality. 1: Skip all cell info. 2: Dump all ports of cell instance. Others: Show error message and ignore it.

Examples

NOTE: The following example uses the syntax for calling the FSDB dumping command in the design. Refer to the syntax section for the correct format for the simulator command line.

```
fsdbDumpMDAByFile("./case1.lst","+fsdbfile+case1.fsdb");
```

where case1.lst contains:

```
[case1.lst]
```

Specify the scope instance:

```
0 top.type_dut.regInst
0 top.type_dut.parameterInst
5 top.type_dut.FunAndTaskInst
+skip_cell_instance=0 0 top.type_dut.tri0Inst
+skip_cell_instance=1 0 top.type_dut.tri1Inst
+skip_cell_instance=2 0 top.type_dut.triInst
3 top.type_dut.enumInst
3 top.type_dut.logicInst
top.type_dut.pstructInst
```

Specify the MDA variable:

```
top.type_dut.mdaInst 0
top.type_dut.mdaInst 0 1 0 2
top.type_dut.mdaInst 0 1 0 2 0 1
+skip_cell_instance=0 top.type_dut.mdaInst 0
+skip_cell_instance=1 top.type_dut.mdaInst 0 1 0 2
```

Variable Example

Specified in the design, the runtime variable can be passed as an argument.

```
...
reg [32*8-1:0] option_reg = "+fsdbfile+novas.fsdb";
```

```
reg [1023:0] suppressFile_reg = "suppress.list"; // reg string
string dumpListFile_string = "dump.list";//string
     ...

fsdbDumpMDAByFile("dump.list", "+fsdbfile+novas.fsdb");
fsdbDumpMDAByFile(dumpListFile_reg);
fsdbDumpMDAByFile(dumpListFile_string, option_reg);
```

Assertion Dumping Command

$fsdbDumpSVA

Description

Dumps the SVA results to the FSDB file. The results of SVA assertions within the specified scopes and/or modules can be dumped and saved to the same FSDB file that contains dumped results for design signals or to a separate file.

Syntax

When specified in the design:

```
$fsdbDumpSVA([ depth, | "level=",depth_var, ]
[instance | "instance=", instance_var ]
[ ,"option" | ,"option=",option_var ]*);

$fsdbDumpSVA;
$fsdbDumpSVA([depth] [, instance] [, "option"]*
```

When specified on the simulator command line:

Synopsys:

```
fsdbDumpSVA [depth] [instance] ["option"]*
```

Cadence:

```
call fsdbDumpSVA [depth] [instance] ["option"]*
```

Modelsim:

```
fsdbDumpSVA
fsdbDumpSVA [depth] [instance] ["option"]*
```

Arguments

`depth`

Specify the levels of hierarchy to dump for the subsequent scopes.

`"level="`

Keyword to identify the next argument is a number.

NOTE: Valid only if the command is specified inside the design.

`depth_var`

Specify how many levels of hierarchy to dump for the subsequent scopes in a variable.

NOTE: Valid only if the command is specified inside the design.

`instance`

Specify the instance name for which all SVA assertions below it are dumped.

`"instance="`

Keyword to identify the content of the next argument is a module scope or signal.

`instance_var`

This definition gives the module scope or signal represented by instance_var that specifies the object to dump.

`option`

+fsdbfile+*filename*	Specify the FSDB file name. If not specified, the default FSDB file name is "*novas.fsdb*".
"+functions"	Enable dumping the SVA in functions in the design. **Example:** *$fsdbDumpSVA("functions");*
"+skip_cell_instance=*mode*"	Enable/Disable dumping of cell instance where *mode* means: 0: Disable functionality. 1 or 2: Skip all cell information. Others: Show error message and ignore it.

`"option="`

Keyword to identify the next argument is a string.

NOTE: Valid only if the command is specified inside the design.

`option_var`

Specify the option in a variable.

NOTE: Valid only if the command is specified inside the design.

Examples

NOTE: The following examples use the syntax for calling the FSDB dumping command in the design. Refer to the syntax section for the correct format for the simulator command line.

```
initial begin
$fsdbDumpvars("+fsdbfile+1.fsdb");
  $fsdbDumpSVA("+fsdbfile+1.fsdb");
  #40 $fsdbDumpoff;
  #40 $fsdbDumpon;
  #10 $fsdbDumpFinish;
  #30 $fsdbDumpfile("2.fsdb");
  $fsdbDumpSVA(1,top);     //dump sva to 2.fsdb
  #43 $finish;
end
```

```
$fsdbDumpSVA;
```

Dump all SVA assertions from design top and its descendents.

```
$fsdbDumpSVA(0, system);
```

Dump all SVA assertions under system and its descendents

```
$fsdbDumpSVA(1, system.arbiter);
```

Dump all SVA assertions under system.arbiter. The dumping level is set to 1, so dumping excludes the SVA assertions under their descendent scopes.

```
$fsdbDumpvarsByFile("scope_lst.txt", "+fsdbfile=hdl.fsdb");
$fsdbDumpSVA(system.arbiter, "+fsdbfile=SVA.fsdb");
```

The first dumping command dumps design signals to the hdl.fsdb file. The second dumping command dumps all SVA assertions under system.arbiter scope and its descendents to the SVA.fsdb file.

NOTE: For VCS users, the **-sverilog** and **-debug_pp** command line options are necessary to dump SVA signals.

NOTE: For ModelSim users, the **-sva** and **-assertdebug** options should be specified in the vsim command line options when dumping assertions. For example,
`>vsim -c top -pli novas_fli.so -sva -assertdebug`

NOTE: To dump vacuous successes correctly with VCS for versions older than VCS 2017.12, specify the **-assert vacuous** option when calling the VCS executable.
The following is a code example for a vacuous success:

```
property P5;
       @(posedge clk)
```

```
                disable iff (rst)
                 (c) |=> (##[0:$] d);
      endproperty : P5
```

Variable Example

If specified in the design, the design variable can be passed as an argument.

```
reg [255:0] depth_reg = 10;
reg [32*8-1:0] option_reg = "+fsdbfile+novas.fsdb";
string option_string = "+fsdbfile+novas.fsdb";
string depth_string = "10";// invalid, number in string format
reg [32*8-1:0] depth_str_reg = "10"; // invalid.
string instance_string = "top";

$fsdbDumpSVA(1, top, "+fsdbfile+novas.fsdb");
$fsdbDumpSVA(1, "option=", option_reg);
$fsdbDumpSVA(top, "option=", option_string);
$fsdbDumpSVA("level=", depth_reg, "+fsdbfile+novas.fsdb");
$fsdbDumpSVA("instance=", instance_string, "option=",
option_reg);
```

NOTE:
1. Immediate assertion (for example, A1: assert(...) ...else $error, A2 assert finial(....)) are dumped in FSDB dumper only for VCS simulators.
2. Sequence (for example, sequence .... endsequence) and property (property...endproperty) is not dumped in FSDB dumper.

$fsdbDumpPSL

Description

Dumps the PSL assert directives in the Verilog or VHDL design to the FSDB file.

NOTE: This command supports the dumper for IUS 13.1, VCS 2015.09, ModelSim 10.2, and their later versions.

NOTE: If the IUS simulator is used, the **+psl_prop** option needs to be in order to dump the PSL properties in the Verilog or VHDL design. This option is not required to be specified in the VCS simulator.

Syntax

```
$fsdbDumpPSL([ depth, | “level=”,depth_var, ]
[instance | “instance=”,instance_var]);
[ ,”option” | ,”option=”,option_var ]*);
```

Arguments

`depth`

Specify the levels of hierarchy to dump for the subsequent scopes. If the subsequent arguments are signals, only these signals are dumped.

`"level="`

Keyword to identify the next argument is a number.

NOTE: Valid only if the command is specified inside the design.

`depth_var`

Specify how many levels of hierarchy to dump for the subsequent scopes in a variable.

NOTE: Valid only if the command is specified inside the design.

`instance`

Specify the instance name for which all SVA assertions below it are dumped.

`"instance="`

Keyword to identify the content of the next argument is a module scope or signal.

`instance_var`

This definition gives the module scope or signal represented by the instance_var that specifies the object to dump. The HDL type should be *string*.

`option`

+fsdbfile+*filename*	Specify the FSDB file name. If not specified, the default FSDB file name is "*novas.fsdb*".
+psl_prop	Dump the PSL properties.

`"option="`

Keyword to identify the next argument is a string.

NOTE: Valid only if the command is specified inside the design.

`option_var`

Specify the option in a variable.

NOTE: Valid only if the command is specified inside the design.

Supported Directive and Verification Unit Types

The *$fsdbDumpPSL* command supports dumping for some directive and verification units.

Table: Supported Directive Types

Directive Type	Support Status (Yes/No)
assert	Yes
assume	Yes
cover	Yes
assume_guarantee	No
restrict_guarantee	No
fairness	No
strong_fairness	No
restrict	No **NOTE:** The restrict directive is not dumped. When the **+psl_prop** option is specified, the property under the restrict directive is created.

The *$fsdbDumpPSL* command supports dumping for the following verification unit types:

- vunit
- vprop
- vmode

Examples

inline

Assume that the file name is *case.v*:

```
module top;
  reg clk = 0;
  always #1 clk = ~clk;
  //psl property p1 = {clk==1;clk==0} @(posedge clk); ---(A)
  //psl assert always {clk==1;clk==0} @(posedge clk); ---(B)
endmodule
```

By default, (B) is dumped. If the **“+fsdb+psl_prop”** option is specified, (A) is also dumped.

vunit / vprop / vmode

Assume that the file name is *case.v.*

```
module top;
    reg clk = 0;
    always #1 clk = ~clk;
endmodule
```

Assume that the file name is *casevunit.v* and this file can only be loaded and compiled with the option:

```
vunit v1 (top) {
    default clock = @(posedge top.clk);
    property p2 = {clk==1;clk==0} @(posedge clk);          ---(C)
    assert always {clk==1;clk==0};                         ---(D)
}
```

By default, (D) is dumped. If the **"+fsdb+psl_prop"** option is specified, (C) is also dumped.

Extra Scope Creation

An extra scope is created when binding the verification unit with the instance, for example:

```
File: unit.v
vunit VUNIT_NAME(top.sub_Inst){
  …
}
File: top.v
module top;
  …
  sub sub_Inst();
endmodule

module sub;
  …
endmodule
```

In this case, an extra scope named *"VUNIT_NAME"* is created under *top.sub_Inst*. The same result is obtained in vprop and vmode.

Naming Rules

When naming the design, the following rules must be followed:

- assert/assume/cover should be keywords when naming the design. For example:
  - top.A1_$assert // (This is an assert.)
  - top.6_case302_v_cover // (This is a cover.)

- The restrict directive is not dumped. When the **+psl_prop** option is specified, the property under the restrict directive is created.
- The cover directive does not have an underlying property.

Unnamed Assertion

[case1]

```
module top;
     reg clk =0;
     reg chk = 1;
     always #1 clk = ~clk;

     //psl property p1 = {clk} @(posedge clk);
     //psl assert always {clk} @(posedge clk);

     initial begin
         $fsdbDumpPSL;
```

If the **+psl_prop** option is not specified, the dumping result is:

```
top.7_case302_v_assert.
```

If the **+psl_prop** option is specified, the dumping result is:

```
top.7_case302_v_assert
top.p1
```

[case2]

```
module top;
    reg clk =0;
    reg chk = 1;
    always #1 clk = ~clk;

    //psl property p1 = {clk} @(posedge clk);
    //psl assert always p1;

    initial begin
        $fsdbDumpPSL;
```

If the **+psl_prop** option is not specified, the dumping result is:

```
top.7_case302_v_assert
```

If the **+psl_prop** option is specified, the dumping result is:

```
top.7_case302_v_assert
top.p1
```

Named Assertion

[case1]

```
module top;
    reg clk =0;
    reg chk = 1;
    always #1 clk = ~clk;

    //psl property p1 = {clk} @(posedge clk);
    //psl A1: assert always {clk} @(posedge clk);

    initial begin
        $fsdbDumpPSL;
```

If the **+psl_prop** option is not specified, the dumping result is:

```
top.A1_$assert
top.A1        // (This module has the same value change as
top.A1_$assert.)
```

If the **+psl_prop** option is specified, the dumping result is:

```
top.A1_$assert
top.A1
top.p1
```

[case2]

```
module top;
    reg clk =0;
    reg chk = 1;
    always #1 clk = ~clk;

    //psl property p1 = {clk} @(posedge clk);
    //psl A1: assert always p1;
      8
    initial begin
        $fsdbDumpPSL;
```

If the **+psl_prop** option is not specified, the dumping result is:

```
top.A1_$assert
top.A1
```

If the **+psl_prop** option is specified, the dumping result is:

```
top.A1_$assert
top.A1
top.p1
```

Unnamed Restrict

```
module top;
    reg clk =0;
```

```
    reg chk = 1;
    always #1 clk = ~clk;

    //psl restrict {clk} @(posedge clk);

    initial begin
        $fsdbDumpPSL;
```

If the **+psl_prop** option is not specified, the dumping result is: none.

If the **+psl_prop** option is specified, the dumping result is:

```
top.__restrict1
```

Named Restrict

```
module top;
    reg clk =0;
    reg chk = 1;
    always #1 clk = ~clk;

    //psl R1: restrict {clk} @(posedge clk);

    initial begin
        $fsdbDumpPSL;
```

If the **+psl_prop** option is not specified, the dumping result is: none.

If the **+psl_prop** option is specified, the dumping result is:

```
top.R1
```

Compilation Tips

The "-assert" option should be specified for PSL execution.

```
> irun –assert design_files -propfile_vlog vlog_vunit_file
-propfile_vhdl vhdl_vunit_file …
```
Bind the vunit in the irun command line.

```
> ncvlog … -assert -propfile vlog_vunit_file
```
Bind the vunit in the ncvlog command line.

```
> ncvhdl … -assert -propfile vhdl_vunit_file
```
Bind the vunit in the ncvhdl command line.

FSDB Dumping Commands Used in VHDL

The Novas object files for FSDB dumping provide the following FSDB dumping commands for VHDL:

fsdbAutoSwitchDumpfile
fsdbDumpfile
fsdbDumpflush
fsdbDumpon, fsdbDumpoff
fsdbDumpvars
fsdbDumpvarsByFile
fsdbDumpvarsES
fsdbDumpFinish
fsdbDumpMDA
fsdbDumpSVA
fsdbDumpVariable
fsdbDumpVariableByFile
fsdbSuppress
fsdbSwitchDumpfile
fsdbDumpfile
fsdbAutoSwitchDumpfile
fsdbDumpFinish
fsdbDumpflush
fsdbDumpVariable
fsdbDumpVariableByFile

Supported Simulators for VHDL FSDB Dumping Commands

The following table illustrates the supported simulator version for each FSDB dumping command:

VHDL Command Name	IUS 13.1 and later	VCSMX 2015.09 and later	ModelSim 10.2 and later
fsdbAutoSwitchDumpfile	x	x	x
fsdbDumpfile	x	x	x
fsdbDumpflush	x	x	x
fsdbDumpoff	x	x	x
fsdbDumpon	x	x	x

fsdbDumpvars	x	x	x
fsdbDumpvarsByFile	x	x	x
fsdbDumpvarsES	x	x	x
fsdbDumpFinish	x	x	x
fsdbDumpMDA*	x*	x*	x*
fsdbDumpSVA	x	x	x
fsdbDumpVariable			x
fsdbDumpVariableByFile			x
fsdbSuppress	x	x	x
fsdbSwitchDumpfile	x	x	x

**fsdbDumpMDA* supports Verilog/SystemVerilog memory or Multi-Dimensional Array (MDA) statements only.

General Dumping Commands

fsdbDumpvars

Description

Dump signals under the specified scopes with the arguments *instance* and *depth*. The signals are saved in the FSDB file created by *fsdbDumpfile* command or the *+fsdbfile+filename* option. If an FSDB file name is not specified, the default file name is *novas.fsdb*.

Syntax

When specified in the design:

```
fsdbDumpvars(depth, “instance”[,“option,*”]);
```

When specified on the simulator command line:

Synopsys:
When specified on the VCS MX UCLI command prompt:

```
fsdbDumpvars [depth] [instance] [option]*
```

Cadence:

```
call fsdbDumpvars
call fsdbDumpvars [depth][instance][option]*
```

Modelsim:

```
fsdbDumpvars
fsdbDumpvars [depth] [instance]
```

NOTE: For VCS users, if there is a Verilog top, use the syntax from the Verilog section.

Arguments

`depth`

Specify how many sub-scope levels under the given scope you want to dump. Specify this argument as 1 to dump the signals under the given scope. Specify this argument as 0 to dump all signals under the given scope and its descendant scopes.

`instance`

Specify the complete path name of the scope you want to dump. Different simulators have different naming rules for describing the complete path name of a scope. You can refer to the simulator documents for complete details.

Assume the VHDL root scope is *root_scope*, *scope1* is the child scope under *root_scope*, and *scope2* is the child scope under *scope1*. In IUS, the complete path name of a scope is*:scope1:scope2*. The left-most colon indicates the root scope and the other colons are used to separate the scope name. In ModelSim, the complete path name of a scope can be either *root_scope.scope1.scope2* or */root_scope/scope1/scope2*.

In VCS MX, the complete path name of a scope is*:root_scope:scope1:scope2*, the left-most colon can be ignored.

`option`

Specify one or more of the following options.

"+IE_Only"	Only IE signals are dumped. This option dumps both the analog and digital signal of all types of IEs such as, `d2a`, `a2d`, `n2e`, `e2n`, `r2e`, and `e2r`.
"+IO_Only"	Only IO port signals are dumped.
"+functions"	Enable dumping of functions in the design. **Example:** *$fsdbDumpvars("+functions");*
"+Reg_Only"	Only reg type signals are dumped.

"+mda"	Dump all memory and MDA signals in all scopes specified in *fsdbDumpvars* option or the entire design if no scope is specified.This option is covered by the "*+all*" option and is effective for SystemVerilog's MDAs only. For VCS users, the VCS option "*+memcbk*" may be needed while compiling the design.
"+packedmda"	Only dump the packed signals in the design (without array and memory types). Refer to the *Supported MDA Format Table* for a summary of the packed or unpacked one-dimensional/ two-dimensional signals that are dumped with or without the *+packedmda* and *+mda* options. **Example:** *$fsdbDumpvars("+packedmda");* **NOTE:** This option replaces the *+mda+packedOnly* option.
+packedmda+struct	Dump the struct or the struct in the packed MDA. **Example:** *$fsdbDumpvars("+packedmda+struct ");*
"+struct"	Dump all structs in all scopes specified in *fsdbDumpvars* option or in the entire design if no scope is specified. This option is covered by the "*+all*" option and is effective for SystemVerilog struct syntax only.
"+skip_cell_instance =*mode*"	Enable/Disable dumping of cell instance where *mode* implies: 0: disable functionality. 1: Skip all cell info. 2: Dump all ports of cell instance. Others: Show error message and ignore it.
"+all"	Dump all signals including the memory, MDA, packed array, structure, union, and packed structure signals in all scopes specified in *fsdbDumpvars* option or the entire design if no scope is specified. For VCS users, the VCS option *"+memcbk"* may be needed when compiling the design.
"+parameter"	Dump parameters. **NOTE:** If the parameter type of Verilog is “scalar”, “real”, or “string”, the name is displayed without range information.
"+trace_process"	Dump VHDL processes.
"+fsdbfile+*filename*"	Specify the FSDB file name. If it is not specified, the default FSDB file name is "*novas.fsdb*".

Supported MDA Format Table

Options	Unpacked 1D for example, T a[ ];	Unpacked 2D for example, T a[ ][ ];	Packed 1D for example, T [ ]a;	Packed 2D for example, T [ ][ ]a;
None			Yes	
"+packedmda"			Yes	Yes
"+mda"	Yes	Yes	Yes	Yes

NOTE: For VCS and IUS users, to allow *fsdbDumpvars* to dump the variables under the processes and the shared variables under the region scopes by default, you can use one of the following methods:
- Add the task option "`+traces_process`" when *fsdbDumpvars* is specified in the design. For example:
```
fsdbDumpvars(0, ":", "+trace_process");
```
- Specify the plus option `+fsdb+trace_process` while running "ncsim" or "simv".
For example:
```
> ncsim work.system:BLK +fsdb+trace_process -tcl
> simv system +fsdb+trace_process
```

- Set the system environment variable *NOVAS_FSDB_TRACE_PROCESS* to a non-zero value before executing the simulation.
For example:
```
> setenv NOVAS_FSDB_TRACE_PROCESS 1
> ncsim work.system:BLK -tcl
```

For ModelSim users, you can use *fsdbDumpVariable* to dump variables. For VCS users, the VCS debug level option needs to be set as `"-debug_all"` for dumping variables under VHDL process.

Examples

NOTE: The following examples use the syntax for calling the command in the design. Refer to the syntax section for the correct format for the simulator command line.

In IUS

```
fsdbDumpvars(0,":");
```

Dump all signals under all scopes of a design.

```
fsdbDumpvars(1,":i_CPU");
```

Only dump the signals under the scope `i_CPU` that is a child scope under the root scope.

```
fsdbDumpvars(0,":",”+fsdbfile+test.fsdb,+mda”);
```

Dump all signals under all scopes to the specified FSDB file, “test.fsdb”. If the design contains SystemVerilog code, also dump MDA signals within the SystemVerilog code under all scopes to the FSDB file.

NOTE: Use a comma to separate options and no space should be left between options.
A colon ":" is suggested for use in the hierarchy expression under a VHDL top design scenario (mixed).

In ModelSim

```
fsdbDumpvars(0,"/system");
```

Dump all signals under the root scope system and its descendant scopes.

```
fsdbDumpvars(1,"/system/i_CPU");
```

Only dump the signals under the scope `i_CPU` that is a child scope under the root scope system.

```
fsdbDumpvars(0,"/system",”+trace_process,+fsdbfile+test.fsdb”);
```

Dump all signals under the root scope to the specified FSDB file, “test.fsdb”. If the design contains SystemVerilog code, also dump all processes within the SystemVerilog code under the root scope to the FSDB file.

NOTE: Use a comma to separate options and no space should be left between options.
A slash "/" or dot "." is suggested for use in the hierarchy expression under a VHDL top design scenario (mixed or pure VHDL).

In VCS MX

```
fsdbDumpvars(0,":system");
```

Dump all signals under the root scope system and its descendant scopes.

```
fsdbDumpvars(1,":system:i_CPU");
```

Only dump the signals under the scope `i_CPU` that is a child scope under the root scope system.

```
fsdbDumpvars(0,":system",”+trace_process,+fsdbfile+test.fsdb”);
```

Dump all signals under system scope and all variables under VHDL processes to the specified FSDB file, “test.fsdb”.

```
fsdbDumpvars("+fsdbfile+my.fsdb")
```

Dump all signals under system scope to the specified FSDB file, "my.fsdb".

NOTE: Use a comma to separate options and no space should be left between options.
A colon ":" is suggested for use in the hierarchy expression under a VHDL top design scenario (mixed or pure VHDL).

The following examples are for mixed-HDL designs. Assume `vlog_inst#` is the Verilog scope name and `vhdl_ent#` is the VHDL scope name. The design hierarchies of 2 theoretical mixed-HDL designs are as follows:

1.

```
vhdl_top --+-- vhdl_ent1 --+-- vhdl_ent2
           |               |
           |               +-- vlog_inst2
           |
           +-- vlog_inst1 -+-- vhdl_ent3
                           |
                           +-- vlog_inst3
```

2.

```
vlog_top --+-- vhdl_ent1 --+-- vhdl_ent2
           |               |
           |               +-- vlog_inst2
           |
           +-- vlog_inst1 -+-- vhdl_ent3
                           |
                           +-- vlog_inst3
```

The following examples are recommended when specifying the path name in *fsdbDumpvars* command to dump a specific scope with simulator dependency delimiter when specified in the design.

In IUS

```
fsdbDumpvars(1,":");
fsdbDumpvars(1,":vhdl_ent1");
fsdbDumpvars(1,":vhdl_ent1:vhdl_ent2");
fsdbDumpvars(1,":vhdl_ent1:vlog_inst2");
fsdbDumpvars(1,":vlog_inst1");
fsdbDumpvars(1,":vlog_inst1.vhdl_ent3");
fsdbDumpvars(1,":vlog_inst1.vlog_ins3");

fsdbDumpvars(1,"vlog_top");
fsdbDumpvars(1,"vlog_top.vhdl_ent1");
fsdbDumpvars(1,"vlog_top.vhdl_ent1:vhdl_ent2");
fsdbDumpvars(1,"vlog_top.vhdl_ent1:vlog_inst2");
fsdbDumpvars(1,"vlog_top.vlog_inst1");
fsdbDumpvars(1,"vlog_top.vlog_inst1.vhdl_ent3");
fsdbDumpvars(1,"vlog_top.vlog_inst1.vlog_ins3");
```

In ModelSim

```
fsdbDumpvars(1,"/vhdl_top");
fsdbDumpvars(1,"/vhdl_top/vhdl_ent1");
fsdbDumpvars(1,"/vhdl_top/vhdl_ent1/vhdl_ent2");
fsdbDumpvars(1,"/vhdl_top/vhdl_ent1/vlog_inst2");
fsdbDumpvars(1,"/vhdl_top/vlog_inst1");
fsdbDumpvars(1,"/vhdl_top/vlog_inst1/vhdl_ent3");
fsdbDumpvars(1,"/vhdl_top/vlog_inst1/vlog_ins3");

fsdbDumpvars(1,"/vlog_top");
fsdbDumpvars(1,"/vlog_top/vhdl_ent1");
fsdbDumpvars(1,"/vlog_top/vhdl_ent1/vhdl_ent2");
fsdbDumpvars(1,"/vlog_top/vhdl_ent1/vlog_inst2");
fsdbDumpvars(1,"/vlog_top/vlog_inst1");
fsdbDumpvars(1,"/vlog_top/vlog_inst1/vhdl_ent3");
fsdbDumpvars(1,"/vlog_top/vlog_inst1/vlog_ins3");
```

Alternatively, you can use a dot (.) as the separator in the scope path names.

```
fsdbDumpvars(1,"vhdl_top");
fsdbDumpvars(1,"vhdl_top.vhdl_ent1");
fsdbDumpvars(1,"vhdl_top.vhdl_ent1.vhdl_ent2");
fsdbDumpvars(1,"vhdl_top.vhdl_ent1.vlog_inst2");
fsdbDumpvars(1,"vhdl_top.vlog_inst1");
fsdbDumpvars(1,"vhdl_top.vlog_inst1.vhdl_ent3");
fsdbDumpvars(1,"vhdl_top.vlog_inst1.vlog_ins3");

fsdbDumpvars(1,"vlog_top");
fsdbDumpvars(1,"vlog_top.vhdl_ent1");
fsdbDumpvars(1,"vlog_top.vhdl_ent1.vhdl_ent2");
fsdbDumpvars(1,"vlog_top.vhdl_ent1.vlog_inst2");
fsdbDumpvars(1,"vlog_top.vlog_inst1");
fsdbDumpvars(1,"vlog_top.vlog_inst1.vhdl_ent3");
fsdbDumpvars(1,"vlog_top.vlog_inst1.vlog_ins3");
```

In VCS MX

```
fsdbDumpvars(1,":vhdl_top");
fsdbDumpvars(1,":vhdl_top:vhdl_ent1");
fsdbDumpvars(1,":vhdl_top:vhdl_ent1:vhdl_ent2");
fsdbDumpvars(1,":vhdl_top:vhdl_ent1:vlog_inst2");
fsdbDumpvars(1,":vhdl_top:vlog_inst1");
fsdbDumpvars(1,":vhdl_top:vlog_inst1:vhdl_ent3");
fsdbDumpvars(1,":vhdl_top:vlog_inst1:vlog_ins3");

fsdbDumpvars(1,":vlog_top");
fsdbDumpvars(1,":vlog_top:vhdl_ent1");
fsdbDumpvars(1,":vlog_top:vhdl_ent1:vhdl_ent2");
fsdbDumpvars(1,":vlog_top:vhdl_ent1:vlog_inst2");
fsdbDumpvars(1,":vlog_top:vlog_inst1");
```

```
fsdbDumpvars(1,":vlog_top:vlog_inst1:vhdl_ent3");
```

NOTE: The rule of delimiters regarding hierarchy expressions is simulator dependent. The above usage is based on the earliest supporting build in simulators, and later builds in simulators may allow more free styles on delimiters or changes without notice.

fsdbDumpvarsES

Description

Dump signal-value-change information to the FSDB file for the signals in the specified Essential Signal list which are within the specified scopes and levels.

LIMITATIONS:
Only hierarchical format for the Essential Signal list is supported. Multiple trees are created in the FSDB file when multiple *fsdbDumpvarsES* system tasks are specified.

Syntax

When specified in the design:

```
fsdbDumpvarsES( depth, “instance”, "es_list_file”[, “option,*”]);
```

NOTE: The first three arguments depth, instance, and es_list_file are required.
You can place a delimiter in the instance argument to represent the design top.
For users of ModelSim, the delimiter between the hierarchical names is slash (/).
For users of IUS or VCS, the delimiter is colon (:).
For example, *fsdbDumpvarsES*(0, ":", myeslist.txt, "+fsdbfile+dump.fsdb").

When specified on the simulator command line:

```
Synopsys: (VCS_MX UCLI command prompt)
fsdbDumpvarsES ESFileName
fsdbDumpvarsES [insatnce] ESFileName
fsdbDumpvarsES [depth] [instance] [option]* ESFileName

Cadence:
call fsdbDumpvarsES ESFileName
call fsdbDumpvarsES [instance] ESFileName
call fsdbDumpvarsES [depth] [instance] [option]* ESFileName

ModelSim:
fsdbDumpvarsES ESFileName
```

```
fsdbDumpvarsES [instance] ESFileName
fsdbDumpvarsES [depth] [instance] [option]* ESFileName
```

Arguments

`depth`

Specify how many levels of hierarchy to dump for the subsequent scopes.

`instance`

This definition gives the module scope or signals that specify the objects to dump.

`es_list_file`

Specify the es list filename.

The file format of the Essential Signal file is as follows:

```
#ESD
# The first line should be "#ESD".
# Used to recognize the file format.

#V2
# The second line should be “#V2”.
# The following is the signal list.
# The first character '<' means the beginning of a scope.
# The scope name is specified sequentially.
<scope_instance_name

# List all signals under the same scope.
signal_instance_name
...
<scope_instance_name
...
>
# The first character '>' indicates move to the next
# higher scope.
...
>

# The following are option values you want to save in FSDB.
# They will be saved under a hidden scope called
# $$ESD_OPTIONS.
# Syntax - $var_name="value"
$op1="system.i_cpu.i_PCU"
# The first character '$' means an option and its value are
# specified sequentially.
```

`option`

Specify one or more of the following options:

+fsdbfile+*filename*	Specify the FSDB file name. If not specified, the default FSDB file name is "*novas.fsdb*".
"+skip_cell_instance=*mode*"	Enable/Disable dumping of cell instance where *mode* implies: 0: Disable functionality. 1 or 2: Skip all cell information. Others: Show error message and ignore it.

Examples

NOTE: The following example uses the syntax for calling the system task in the design. Refer to the syntax section for the correct format for the simulator command line.

```
fsdbDumpvarsES(0, ":", "myeslist.list");
```

Dumps all signals in the Essential Signal list to the FSDB. Hierarchical format is used in myeslist.list.

```
fsdbDumpvarsES(1, "system.i_CPU.i_ALUB", "+fsdbfile+target.fsdb"
, "myeslist.list")
```

Dumps all signals in the Essential Signal file (myeslist.list) which are also within the specified scope(s) and level to *target.fsdb*. The immediate child scope(s) of the specified scope(s) and level are identified. If the child scope is a library cell instance, it is ignored. For other immediate child scopes, the external nets of output boundary signals are dumped.

fsdbDumpMDA

Description

Dump value changes of SystemVerilog MDA (one or multi-dimensional array) signals to the FSDB file.

NOTE: *fsdbDumpMDA* supports MDA statements that are specified in the Verilog/SystemVerilog portion of a mixed language design only. For pure VHDL arrays, which can be dumped by *fsdbDumpvars*, there is no specific command for performing this action.

fsdbDumpMDA needs to be added in the initial block. Invoking *fsdbDumpMDA* multiple times in the source code, for example specifying inside the always block, may cause the FSDB size to be increased, since *fsdbDumpMDA* traverses the design each time it is invoked and saves the hierarchy tree into the FSDB file.

Syntax

When specified in the design:

```
fsdbDumpMDA( depth, "instance" [,"option,*"] );
```

NOTE: Inside quotation marks, use a comma to separate options and no space should be left between options.

The first two arguments *depth* and *instance* are required. You can place a delimiter in the *instance* argument to represent the design top. For example, *fsdbDumpMDA*(0, ":"). For ModelSim users, the delimiter between the hierarchical names is slash (/).

When specified on the simulator command line:

```
Synopsys: (VCS_MX UCLI command prompt)
fsdbDumpMDA [depth] [instance]
fsdbDumpMDA [depth] [instance] [option]*

Cadence:
call fsdbDumpMDA [depth] [instance]
call fsdbDumpMDA [depth] [instance] [option]*

Modelsim:
fsdbDumpMDA [depth] [instance]
fsdbDumpMDA [depth] [instance] [option]*
```

Arguments

`depth`

Specify how many levels of hierarchy to dump for the subsequent scopes.

`instance`

This definition gives the instance/signal that contains the MDA objects you want to dump.

`option`

+fsdbfile+*filename*	Specify the FSDB file name. If not specified, the default FSDB file name is "*novas.fsdb*".
"+skip_cell_instance=*mode*"	Enable/Disable dumping of cell instance where *mode* implies: 0: Disable functionality. 1 or 2: Skip all cell information. Others: Show error message and ignore it.

Examples

Assume the following memory and MDA signal are defined in SystemVerilog design part of the mixed HDL design:

```
reg [7:0] plane [15:0];
reg [7:0] screen [79:0][0:24][1:0];
```

The following examples are specified in a VHDL design.

```
fsdbDumpMDA(0, “plane” );
```

Dump all cells of memory *plane*.

```
fsdbDumpMDA(0, “system”);
```

Dump all memories and MDAs under the *system* scope and its child scopes.

```
fsdbDumpMDA(1, “system.i_pram”);
```

Dump all memories and MDAs under the *system.i_pram* scope.

```
fsdbDumpMDA(0, top, “+skip_cell_instance=1,+fsdbfile+dump.fsdb”);
```

Dump all memories and MDAs under *top* scope and skip cell instance.

fsdbDumpSVA

Description

Dump SystemVerilog Assertion (SVA) results to the FSDB file. The results of SVA within the specified scopes and/or modules can be dumped and saved to the same FSDB file that contains dumped results for design signals or to a separate file.

Syntax

When specified in the design:

```
fsdbDumpSVA( depth, "instance" [,"option,*"] );
fsdbDumpSVA( depth, ":" ,"+fsdbfile=filename");
```

NOTE: The first two arguments *depth* and *instance* are required. You can place a delimiter in the *instance* argument to represent the design top. For example, “:”. For the users of ModelSim, the delimiter between the hierarchical names is slash (/).

When specified on the simulator command line:

```
Synopsys: (VCS_MX UCLI command prompt)
fsdbDumpSVA
fsdbDumpSVA [depth] [instance] [option]*

Cadence:
call fsdbDumpSVA
call fsdbDumpSVA [depth] [instance] [option]*

Modelsim:
fsdbDumpSVA
```

```
fsdbDumpSVA [depth] [instance][option]*
```

Arguments

`depth`

Specify the levels of hierarchy to dump for the subsequent scopes.

`instance`

Specify the instance name for dumping all SVA assertions below it.

`option`

+fsdbfile+*filename*	Specify the FSDB file name. If not specified, the default FSDB file name is "*novas.fsdb*".
"+skip_cell_instance=*mode*"	Enable/Disable dumping of cell instance where *mode* implies: 0: Disable functionality. 1 or 2: Skip all cell information. Others: Show error message and ignore it.

NOTE: For VCS users, the **-sverilog** and **-debug_pp** command line options are necessary to dump SVA signals.

For ModelSim users, the **-sva** and **-assertdebug** options should be specified in the vsim command line options when dumping assertions. For example,

```
> vsim -c top -pli novas_fli.so -sva -assertdebug
```

To dump vacuous successes correctly with VCS for previous versions of VCS 2017.12, specify the **-assert vacuous** option when calling the VCS executable.
The following is a code example for a vacuous success:

```
property P5;
       @(posedge clk)
        disable iff (rst)
         (c) |=> (##[0:$] d);
endproperty : P5
```

Examples

NOTE: The following examples use the syntax for calling the FSDB dumping command in the design. Refer to the syntax section for the correct format for the simulator command line.

```
process begin
fsdbDumpvars(0, “top”, “+fsdbfile+my.fsdb”);
  fsdbDumpSVA(0, “top”);
```

```
  wait for 5 ns;
  fsdbDumpoff;
  wait for 5 ns;
  fsdbDumpon;
  wait for 5 ns;
  fsdbDumpFinish;
  fsdbDumpfile("2.fsdb");
  fsdbDumpSVA(1, “top”);     --dump sva to 2.fsdb
  wait for 5 ns;
  wait;
end process;

fsdbDumpSVA(0, “:”);
```

Dump all SVA assertions from design top and its descendent's.

```
fsdbDumpSVA(0, “system”);
```

Dump all SVA assertions under system and its descendent's.

```
fsdbDumpSVA(1, “system.arbiter”);
```

Dump all SVA assertions under *system.arbiter*. The dumping level is set to 1, so dumping excludes the SVA assertions under their descendent scopes.

```
fsdbDumpvarsByFile(“scope_lst.txt”, “+fsdbfile=hdl.fsdb”);
fsdbDumpSVA(0, “system.arbiter”, “+fsdbfile=SVA.fsdb”);
```

The first dumping command dumps design signals to the *hdl.fsdb* file. The second dumping command dumps all SVA assertions under *system.arbiter* scope and its descendent's to the *SVA.fsdb* file.

NOTE: 1. Immediate assertion (for example, A1: assert(….) ...else `$error`, A2 assert finial(…)) are not dumped in FSDB Dumper.
2. Sequences (for example, sequence … endsequence) and properties (property…endproperty) are not dumped by the FSDB dumper.

---

fsdbDumpvarsByFile

Description

Dump several scopes and signals defined in a text file to the designated FSDB file. *fsdbDumpvarsByFile* can be invoked more than once to dump several scopes/signals defined in different text files to specific FSDB files. There are two schemes to specify designated FSDB files as follows:

1. No FSDB file name specified in *fsdbDumpvarsByFile* (only one argument is required). You may use *$fsdbDumpfile* or use argument “*+fsdbfile+filename.fsdb*” (takes another argument) to specify the

designated FSDB file. If there is no FSDB file specified, *novas.fsdb* is taken as the default file name.

2. Specify FSDB file name in *fsdbDumpvarsByFile* (two arguments needed). You must use different FSDB files for each call.

NOTE: *fsdbDumpvarsByFile* does not support text files in Essential Signal Dumping (ESD) file format. Alternately, you can use *fsdbDumpvarsES* dumping command.

Syntax

When specified in the design:

```
fsdbDumpvarsByFile ( "textFileName" [,"+fsdbfile+filename"] );
```

When specified on the simulator command line:

Synopsys:
```
fsdbDumpvarsByFile textFileName [+fsdbfile+filename]
```

Cadence:
```
call fsdbDumpvarsByFile textFileName [+fsdbfile+filename]
```

ModelSim:
```
fsdbDumpvarsByFile textFileName [+fsdbfile+filename]
```

Arguments

`"textFileName"`

Specify the name of the text file containing the scopes to dump (in plain text format).

`option`

+fsdbfile+*filename*	Specify the FSDB file name. If not specified, the default FSDB file name is "*novas.fsdb*".

File Format of the Text File

```
# comment
[option_in_file]* [depth] instance_name
```

The options are the same as *fsdbDumpvars* (repeated below for convenience). Options in the following table are for the text file only. They can only be specified in the [option_in_file] location of the "textfile" only. For example:

```
+all 1 system
+mda +struct 3 system.i_cpu
```

"+IO_Only"	Only IO port signals are dumped.
"+functions"	Enable dumping of functions in the design.
"+Reg_Only"	Only reg type signals are dumped.
"+mda"	Dump all memory and MDA signals in all scopes specified in *fsdbDumpvars* or the entire design if no scope is specified. This option is covered by the *"+all"* option and is effective for SystemVerilog's MDAs only. For VCS users, the VCS option *"+memcbk"* may be needed when compiling the design.
"+packedmda"	Only dump the packed signals in the design (without array and memory types). The *Supported MDA Format Table* summarizes if the packed or unpacked one-dimensional/two-dimensional signals are dumped with or without the *+packedmda* and *+mda* options. **Example:** *$fsdbDumpvars("+packedmda");* **NOTE:** This option replaces the *+mda+packedOnly* option.
+packedmda+struct	Dump the struct or the struct in packed MDA. **Example:** *$fsdbDumpvars("+packedmda+struct ");*
"+struct"	Dump all structs in all scopes specified in *fsdbDumpvars* or in the entire design if no scope is specified. This option is covered by the "*+all*" option and is effective for SystemVerilog struct syntax only.
"+skip_cell_instance=*mode*"	Enable/Disable dumping of cell instance where *mode* implies: 0: Disable functionality. 1: Skip all cell info. 2: Dump all ports of cell instance. Others: Show error message and ignore it.
"+all"	Dump all signals including the memory, MDA, packed array, structure, union and packed structure signals in all scopes specified in *fsdbDumpvars* or the entire design if no scope is specified. For VCS users, the VCS option *"+memcbk"* may be needed when compiling the design.
"+parameter"	Dump parameters. **NOTE:** If the parameter type of Verilog is “scalar”, “real”, or “string”, the name is displayed without range information.
"+trace_process"	Dump VHDL process.

Supported MDA Format Table

Options	Unpacked 1D for example, T a[ ];	Unpacked 2D for example, T a[ ][ ];	Packed 1D for example, T [ ]a;	Packed 2D for example, T [ ][ ]a;
None			Yes	
"+packedmda"			Yes	Yes
"+mda"	Yes	Yes	Yes	Yes

Examples

```
fsdbDumpvarsByFile("./case1.lst","+fsdbfile+case1.fsdb");

[case1.lst]
+Reg_Only 0 top.type_dut.regInst
+parameter 0 top.type_dut.parameterInst
+functions 5 top.type_dut.FunAndTaskInst
+skip_cell_instance=1 0 top.type_dut.tri0Inst
+skip_cell_instance=1 +mda 0 top.type_dut.tri1Inst
+skip_cell_instance=1 +packedmda 0 top.type_dut.triInst
+mda 3 top.type_dut.enumInst
+packedmda 3 top.type_dut.logicInst
+struct top.type_dut.pstructInst
```

fsdbSuppress

Description

Specify the scopes that is not dumped by the *fsdbDumpvars*, *fsdbDumpvarsByFile*, *fsdbDumpMDA* or *fsdbDumpVariable* commands. *fsdbSuppress* supports suppressing by file name or the target scopes passed as arguments. The supported target scope can be modules, instances, scopes, or signals in complete hierarchical format. Each format is called in different *fsdbSuppress* calls respectively.

NOTE: *fsdbSuppress* should be specified before *fsdbDumpvars*, *fsdbDumpvarsByFile*, *fsdbDumpMDA*, or *fsdbDumpVariable*; if not, *fsdbSuppress* does not work.

The setting of *fsdbSuppress* are reset after *fsdbDumpFinish* is executed.

Syntax

When specified in the design:

```
fsdbSuppress( "suppress_file" | "instance,*");
```

```
fsdbSuppress("suppress_file");
fsdbSuppress("instance,*");
```

NOTE: Use a comma to separate instances or signal_names and no space should be left between the instances or signal_names.

When specified on the simulator command line:

Synopsys:
```
fsdbSuppress suppress_file
fsdbSuppress instance*
```

Cadence:
```
call fsdbSuppress suppress_file
call fsdbSuppress instance*
```

ModelSim:
```
fsdbSuppress suppress_file
fsdbSuppress instance*
```

Arguments

`suppress_file`

Specify the file containing the names of the scopes that are not dumped.

`“instance”`

Specify one or more scopes that are not dumped.

`“signal_name”`

Specify the signal in full hierarchical format that are not dumped.

NOTE: All supported suppression methods should be used separately for each *fsdbSuppress* called; cannot combine multiple suppression types in one *fsdbSuppress* command.

For file syntax, each line of the file is the complete name of a scope that you do not want to be dumped.

Examples

Example 1

```
fsdbSuppress(dump_top,system.i_pram);
fsdbDumpvars(0,dump_top);
```

Example 2

```
fsdbSuppress(“./suppress.lst”);

[suppress.lst]
```

```
system.R_W
system.i_cpu.C0
system.i_cpu.C1
system.i_cpu.C5
system.i_cpu.C6
...
```

NOTE: Both target scopes specified as task parameters or listed in suppression file supports instances and modules format.

fsdbDumpon, fsdbDumpoff

Description

These FSDB dumping commands turn dumping *on* and *off*. *fsdbDumpon/ fsdbDumpoff* has the highest priority and overrides all other FSDB dumping commands such as *fsdbDumpvars*, *fsdbDumpvarsES*, *fsdbDumpMDA*, *fsdbDumpSVA*, *fsdbDumpvarsByFile,* or *fsdbDumpVariable*.

NOTE: *fsdbDumpon*/*fsdbDumpoff* is not restricted to only *fsdbDumpvars*. If there is more than one FSDB file open for dumping at one simulation run, *fsdbDumpon*/*fsdbDumpoff* may only affect a specific FSDB file by specifying the specific file name.

Syntax

When specified in the design:

```
fsdbDumpon( ["+fsdbfile+filename"] );
fsdbDumpoff( ["+fsdbfile+filename"] );
```

When specified on the simulator command line:

```
Synopsys: (VCS_MX UCLI command prompt)
fsdbDumpon [+fsdbfile+filename]
fsdbDumpoff [+fsdbfile+filename]

Cadence:
call fsdbDumpon [+fsdbfile+filename]
call fsdbDumpoff [+fsdbfile+filename]

ModelSim:
fsdbDumpon [+fsdbfile+filename]
fsdbDumpoff [+fsdbfile+filename]
```

Arguments

`option`

+fsdbfile+*filename*	Specify the FSDB file name. If not specified, the default FSDB file name is "*novas.fsdb*".

Examples

NOTE: The following example uses the syntax for calling the FSDB dumping command in the design. Refer to the syntax section for the correct format for the simulator command line.

```
process begin
        wait for 5 ns;
    fsdbDumpvars(0, "top");
    wait for 5 ns;
    fsdbDumpoff;
    wait for 5 ns;
    fsdbDumpon;
    wait for 5 ns;
    wait;
end process;
```

The value changes for all variables in this design example are dumped from time 5ns to time 10ns and from time 15ns to time 20ns.

fsdbSwitchDumpfile

Description

Support multiple FSDB files during a simulation run by specifying the currently active FSDB file name. When this command is executed, the previously opened FSDB file is closed and the new FSDB file is created to continue saving the values change data and act as current active FSDB file.

Since the FSDB dumper supports more then one open file for saving data simultaneously, the source file that is supposed to be switched can be specified by the "+*fsdbfile+sourcefilename*" option, if not the source file must be the currently active FSDB file.

Syntax

When specified in the design:

```
fsdbSwitchDumpfile("NewFileName" [ ,"+fsdbfile+src_file"] );

fsdbSwitchDumpfile("NewFileName");
fsdbSwitchDumpfile("NewFileName", "+fsdbfile+src_file");
```

When specified on the simulator command line:

Synopsys and ModelSim:
```
fsdbSwitchDumpfile NewFileName
fsdbSwitchDumpfile NewFileName +fsdbfile+src_file
```

Cadence:
```
call fsdbSwitchDumpfile NewFileName
call fsdbSwitchDumpfile NewFileName +fsdbfile+src_file
```

Arguments

`NewFileName`

The target new name of the FSDB file generating by the Novas object files for FSDB dumping.

`Option`

+fsdbfile+src_file	Specify the source FSDB file to be switched. (include the *.fsdb* file extension in the name). This is optional; if it is not specified, the FSDB dumper switches dump finish on all active FSDB files.

Examples

NOTE: The following example uses the syntax for calling the FSDB dumping command in the design. Refer to the syntax section for the correct format for the simulator command line.

```
fsdbDumpfile("test0.fsdb");
fsdbDumpvars(0, “system”);
fsdbDumpvars(3, “system”, “+fsdbfile+f1.fsdb);
wait for 100 ns
fsdbSwitchDumpfile(“test1.fsdb”, “+fsdbfile+test0.fsdb”);
wait for 100 ns
fsdbSwitchDumpfile("test2.fsdb", “+fsdbfile+f1.fsdb”);
```

Create *test0.fsdb* to dump the design initially. Create `f1.fsdb` to dump different depth of design. Switch `test0.fsdb` to `test1.fsdb` at the simulation time 100 ns. Then switch `f1.fsdb` to `test2.fsdb` at the simulation time 200 ns.

fsdbDumpfile

Description

Specify the FSDB file name created by the Novas object files for FSDB dumping. If it is not specified, the default FSDB file name is "*novas.fsdb*". This

command is valid only before *fsdbDumpvars* is executed and is ignored if specified after *fsdbDumpvars*.

To restrict the largest dump-file size to the user-defined size limitation, specify the limit size in megabyte format. This function affects the FSDB file that is specified in "FSDB_Name". The FSDB dumper uses the sliding window scheme to keep the last signal values in the FSDB file. It drops the old values if the file size exceeds the user-specified limitation.

For *fsdbDumpvars*, *fsdbDumpMDA*, *fsdbDumpvarsES*, *fsdbDumpSVA*, and *fsdbDumpvarsByFile*, you can use the "*+fsdbfile+filename.fsdb*" option to specify the target FSDB file to dump. It is similar to using *fsdbDumpfile* then *fsdbDumpvars*.

NOTE: To leverage the memory consumption, the maximum number of opened FSDB files for dumping in one simulation run should not be more than 32 files.

Syntax

When specified in the design:

```
fsdbDumpfile( "FSDB_Name" [ ,limit_size] );
```

When specified on the simulator command line:

```
Synopsys: (VCS_MX UCLI command prompt)
fsdbDumpfile FSDB_Name [limit_size]

Cadence:
call fsdbDumpfile FSDB_Name [limit_size]

ModelSim:
fsdbDumpfile FSDB_Name [limit_size]
```

Arguments

`FSDB_Name`

Specify the name of the FSDB file generated by the Novas object files for FSDB dumping.

`limit_size`

Specify the maximum FSDB file size in megabyte.

NOTE: The minimum file size of the FSDB file is 10M. If you set the size less than 10M, it is set to 10M instead.

Examples

NOTE: The following example uses the syntax for calling the FSDB dumping command in the design. Refer to the syntax section for the correct format for the simulator command line.

```
fsdbDumpfile("novas.fsdb", 32);
  /* dumpfile novas.fsdb, limit 32 megabyte */
fsdbDumpvars(0, “design_top”);
```

fsdbAutoSwitchDumpfile

Description

Automatically switch to a new dump file when the working FSDB file hits the specified size limitation. The unit associated with the *fsdbAutoSwitchDumpfile File_Size* parameter is a megabyte.

Syntax

When specified in the design:

```
fsdbAutoSwitchDumpfile( File_Size, "FSDB_Name", Number_of_Files
[,"log_filename"] );
```

When specified on the simulator command line:

Synopsys:
```
fsdbAutoSwitchDumpfile File_Size FSDB_name Number_of_Files
[log_filename]
```

Cadence:
```
call fsdbAutoSwitchDumpfile File_Size FSDB_name Number_of_Files
[log_filename]
```

ModelSim:
```
fsdbAutoSwitchDumpfile File_Size FSDB_name Number_of_Files
[log_filename]
```

Arguments

`File_Size`

The FSDB file size limit in MB.

NOTE: The minimum file size of the FSDB file is 10MB. If you set the size less than 10MB, it is set to 10MB instead.

`FSDB_name`

FSDB file generating by the Novas object files for FSDB dumping.

`Number_of_Files`

The maximum number of FSDB files you want to be created.

NOTE: If the number is put as 0, the FSDB dumper creates new FSDB files without file number limitation.

`log_filename`

This argument is optional. You can specify the filename for the log file.

Examples

NOTE: The following examples use the syntax for calling the FSDB dumping command in the VDHL design. Refer to the syntax section for the correct format for the simulator command line.

```
process
begin
fsdbAutoSwitchDumpfile(10, "test.fsdb", 20);
fsdbDumpvars(0, ":system");
wait;
end process;
```

The first FSDB file is named as *test_000.fsdb*. The second FSDB file *test_001.fsdb* is created when the file size of *test_000.fsdb* reaches near 10MB. The third FSDB file *test_002.fsdb* is created when the file size of *test_001.fsdb* reaches near 10MB, and so on. The first FSDB file *test_000.fsdb* is overwritten when the file size of *test_019.fsdb* reaches near 10MB. The default *test.log* file lists the time range of each FSDB file.

```
process begin
fsdbAutoSwitchDumpfile(10, "test.fsdb", 20, "my.log");
fsdbDumpvars(0, "system");
wait;
end process;
```

The number and the maximum file size of the FSDB file are the same as the previous example, but the log file is set to *my.log*.

fsdbDumpFinish

Description

Close all FSDB files in the current simulation and stop dumping the signals. Although all FSDB files are closed automatically at the simulation finish, you can invoke this dumping command to explicitly close the FSDB files during the simulation. You can create a new FSDB file after you have closed the FSDB file. You cannot append the new dumping data to a closed FSDB file due to the current limitation of the FSDB file architecture. If you try to open the same

filename of FSDB file again during the same simulation run, the new FSDB file is renamed automatically.

Syntax

When specified in the design:

```
fsdbDumpFinish;
```

When specified on the simulator command line:

Synopsys and **ModelSim**:
```
fsdbDumpFinish
```

Cadence:
```
call fsdbDumpFinish
```

Examples

NOTE: The following examples use the syntax for calling the commands in the design. Refer to the syntax section for the correct format for the simulator command line.

Example 1

```
fsdbDumpfile("CPU.fsdb");
fsdbDumpvars(0, ":"); -- IUS
-- fsdbDumpvars(0, "system"); -- ModelSim
-- fsdbDumpvars(0, ":system"); -- VCS-MX
wait for 2000 ns;
fsdbDumpFinish;
```

Close the *CPU.fsdb* file at 2000 ns.

Example 2

```
fsdbDumpfile("CPU.fsdb");
fsdbDumpvars(0, ":"); -- IUS
-- fsdbDumpvars(0, "system"); -- ModelSim
-- fsdbDumpvars(0, ":system"); -- VCS-MX
wait for 2000 ns;
fsdbDumpFinish;
wait for 2000 ns;
fsdbDumpfile("CPU.fsdb");
fsdbDumpvars(0, ":i_CPU"); -- IUS
-- fsdbDumpvars(0, "system/i_CPU"); -- ModelSim
-- fsdbDumpvars(0, ":system:i_CPU"); -- VCS-MX
wait;
```

Close the *CPU.fsdb* file at 2000 ns. Then open a new FSDB file at 4000 ns. Because, the same FSDB file is reopened, this new FSDB file is renamed as

CPU_r000.fsdb. Only *system.i_CPU* and its child regions are dumped to *CPU_r000.fsdb*.

fsdbDumpflush

Description

This FSDB dumping command may be used in the HDL code and also through the command line for interactive simulation control. The command forces the signal values to be flushed to the FSDB file while the simulation is running. This allows you to check the current simulation results whenever it is ready instead of waiting for the simulation to end or the internal buffer to be filled.

NOTE: The Novas object files for FSDB dumping automatically flushes the value change data to the FSDB file on disk under the following conditions:
- The simulation stops due to a `$stop` statement in Verilog design.
- The simulation stops at the specified breakpoint.
- The Ctrl-C is used to break the simulation.

Syntax

When specified in the design:

```
fsdbDumpflush;
```

When specified on the simulator command line:

Synopsys and ModelSim:
```
fsdbDumpflush
```

Cadence:
```
call fsdbDumpflush
```

fsdbDumpVariable

Description

This dumping command dumps the specified variable value changes to the file created by *fsdbDumpfile* or by *+fsdbfile+filename* option. It is valid only in ModelSim. The variable name must be specified using its complete hierarchical path. Refer to the ModelTech definitions for the full syntax. For example, the delimiter of the full hierarchy path in ModelSim should be “/” or “.”.

NOTE: The FSDB dumper only flushes the current value of variables into the FSDB file as MTI does not currently support a value change callback for variables. To generate variable information throughout the

simulation, specify *fsdbDumpVariable* in a process, not at the *vsim* prompt. For example:

```
P1: process(...)
...
begin
fsdbDumpVariable(...);
...
end process P1;
```

Syntax

When specified in the design:

```
fsdbDumpVariable("variableName" [, “+fsdbfile+filename”]);
```

When specified on the ModelSim simulator command line:

```
fsdbDumpVariable variableName [+fsdbfile+filename]
```

Arguments

`variableName`

Specify the variable name in the complete hierarchy format whose value changes should be dumped to the FSDB file.

`option`

+fsdbfile+*filename*	Specify the FSDB file name. If it is not specified, FSDB dumper dumps the result into the default file.

Examples

NOTE: The following example uses the syntax for calling the commands in the design. Refer to the syntax section for the correct format for the simulator command line.

```
fsdbDumpVariable(“/tb/File_dump/a_complex_rec”);
```

Dump the *tb:File_dump:a_complex_rec* variable data into FSDB file.

fsdbDumpVariableByFile

Description

Dump several variables defined in a text file to the file created by *fsdbDumpfile*. It is valid only in ModelSim.

NOTE: The FSDB dumper only flushes the current value of variables into the FSDB file as MTI does not currently support a value change callback for variables. To generate variable information throughout the simulation, use *fsdbDumpVariableByFile* command in a process, not at the vsim prompt. For example:

```
P1: process(...)
...
begin
fsdbDumpVariableByFile(...);
...
end process P1;
```

Syntax

When specified in the design:

```
fsdbDumpVariableByFile("txt_filename" [, "+fsdbfile+filename"]);
```

When specified on the ModelSim simulator command line:

```
fsdbDumpVariableByFile txt_filename [+fsdbfile+filename]
```

Arguments

`txt_filename`

Specify the text file containing variable name whose value changes should be dumped to the FSDB file.

The file format of the text file is as follows:

```
# comment
variable_instance_name
variable_instance_name
...
```

The variable name must be specified using its complete hierarchical path. Refer to the ModelTech definitions for the full syntax.

`option`

+fsdbfile+*filename*	Specify the FSDB file name. If it is not specified, FSDB dumper dumps the result into the default file.

NOTE: Specify the complete hierarchy path of the variables with ModelSim VHDL delimiters “/” or “.”. For example, /system/i_pram2/pram2/innermem or system.i_pram2.pram2.innermem.

Examples

NOTE: The following example uses the syntax for calling the commands in the design. Refer to the syntax section for the correct format for the simulator command line.

```
fsdbDumpVariableByFile(“dump.lst”, “+fsdbfile+dump.fsdb”);
```

To dump variable instances defined in *dump.lst* file into *dump.fsdb* file. The *dump.lst* file contains the following text:

```
# For ModelSim
/system/i_pram2/pram2/innermem
/system/i_pram2/pram2/memaddr
```

Linking with Synopsys Simulators

NOTE: A `novas_dump.log` file gets created during simulation to record some essential simulation/dumping information, such as simulation options used, environment variable settings, and Novas object directories linked. This log file is needed to report a dumper related issue effectively.

VCS

Set the VCS_HOME environment variable.

```
% setenv VCS_HOME $VCS_INST_DIR
```

Before running the simulation (`simv`), the `VERDI_HOME` environment variable must be set and point to the directory where the Verdi application is installed. For example:

```
% setenv VERDI_HOME $VERDI_INST_DIR
```

Compile the design with the following options:

`-debug_access+pp` or `-debug_access` or `-deubug_access+all`

The correct version of the FSDB dumper is automatically linked regardless of the VCS version you are running.

The VCS option, `-debug_access+pp`, is an aggregation of `+vpi`, `+vcsd`, and `+memcbk`, and it contains the minimal required debug capability for post-processing debug's need. For other VCS options in different debug level capabilities, refer to VCS' documentation or *Appendix B: Required VCS MX Options for Linking With FSDB Dumper* for more details.

Example of Linux 32bit Platforms

```
% vcs -debug_access+pp design.v
```

Linking with VCS using VHDL Dumping Commands

If you want to invoke any FSDB dumping command in VHDL, or any FSDB foreign functions in the VHDL code, you must compile and include a VHDL package in your design.

Use the following steps:

1. Use VCS MX analyzer, `vhdlan`, to compile the Verdi-provided VHDL file to the same directory as the design is saved.

   ```
   $VERDI_HOME/share/PLI/VCS/${PLATFORM}/novas.vhd
   % vhdlan ${NOVAS_HOME}/share/PLI/VCS/${PLATFORM}/novas.vhd
   ```

   The VHDL file `novas.vhd` contains the definitions of the FSDB foreign functions.

2. Use the novas package in any VHDL design file that invokes FSDB foreign functions. For example:

   ```
   library IEEE;
   use IEEE.std_1364.all;
   use work.novas.all; --using novas package.
   entity testbench is
   end;
   architecture blk testbench is
   Begin
   ...
   Process begin: dump
   fsdbDumpvars(0, ":", "+fsdbfile+signal.fsdb"); -- call
   VHDL procedure
   wait;
   end process
   end;
   ```

3. Recompile the modified VHDL files.

   VCS MX loads the Novas object files for FSDB dumping during the simulation.

   To learn more about the usage of the FSDB foreign functions, refer to the *FSDB Dumping Commands Used in VHDL* section.

VCS/VCS MX (VCS/VCS MX 2017.03, 2017.12, and 2018.09) - SystemC and HDL (Verilog/ SystemVerilog, VHDL) Mixed

SystemC Dumper for Synopsys VCS/VCS MX 2017.03 / SystemC 2.2.0

Platform	Compiler	Library Directory
Linux (32-bit)	GNU GCC- 4.2.2 / 4.5.2/ 4.7.2/ 4.8.3/ 5.2.0	share/PLI/VCS/LINUX
Linux (64-bit)	GNU GCC- 4.2.2 / 4.5.2/ 4.7.2/ 4.8.3/ 5.2.0	share/PLI/VCS/LINUX64
SUSE (32-bit)	GNU GCC- 4.2.2 / 4.5.2/ 4.7.2/ 4.8.3/ 5.2.0	share/PLI/VCS/SUSE32
SUSE (64-bit)	GNU GCC- 4.2.2 / 4.5.2/ 4.7.2/ 4.8.3/ 5.2.0	share/PLI/VCS/SUSE64

SystemC Dumper for Synopsys VCS/VCS MX 2017.03 / SystemC 2.3.0

Platform	Compiler	Library Directory
Linux (32-bit)	GNU GCC- 4.5.2/ 4.7.2/4.8.3/ 5.2.0	share/PLI/VCS/LINUX
Linux (64-bit)	GNU GCC- 4.5.2/ 4.7.2/4.8.3/ 5.2.0	share/PLI/VCS/LINUX64
SUSE (32-bit)	GNU GCC- 4.5.2/ 4.7.2/4.8.3/ 5.2.0	share/PLI/VCS/SUSE32
SUSE (64-bit)	GNU GCC- 4.5.2/ 4.7.2/4.8.3/ 5.2.0	share/PLI/VCS/SUSE64

SystemC Dumper for Synopsys VCS/VCS MX 2017.03 / SystemC 2.3.1

Platform	Compiler	Library Directory
Linux (32-bit)	GNU GCC- 4.7.2 / 4.8.3 / 5.2.0 / 6.2.0	share/PLI/VCS/LINUX
Linux (64-bit)	GNU GCC-4.7.2 / 4.8.3 / 5.2.0 / 6.2.0	share/PLI/VCS/LINUX64

SUSE (32-bit)	GNU GCC-4.7.2 / 4.8.3 / 5.2.0 / 6.2.0	share/PLI/VCS/SUSE32
SUSE (64-bit)	GNU GCC-4.7.2 / 4.8.3 / 5.2.0 / 6.2.0	share/PLI/VCS/SUSE64

SystemC Dumper for Synopsys VCS/VCS MX 2017.12 / SystemC 2.2.0

Platform	Compiler	Library Directory
Linux (32-bit)	GNU GCC-4.2.2/ 4.3.2/ 4.7.2 / 4.8.3 / 5.2.0	share/PLI/VCS/LINUX
Linux (64-bit)	GNU GCC-4.2.2/ 4.3.2/ 4.7.2 / 4.8.3 / 5.2.0	share/PLI/VCS/LINUX64
SUSE (32-bit)	GNU GCC-4.2.2/ 4.3.2/ 4.7.2 / 4.8.3 / 5.2.0	share/PLI/VCS/SUSE32
SUSE (64-bit)	GNU GCC-4.2.2/ 4.3.2/ 4.7.2 / 4.8.3 / 5.2.0	share/PLI/VCS/SUSE64

SystemC Dumper for Synopsys VCS/VCS MX 2017.12 / SystemC 2.3.0

Platform	Compiler	Library Directory
Linux (32-bit)	GNU GCC- 4.5.2/ 4.7.2/4.8.3/ 5.2.0	share/PLI/VCS/LINUX
Linux (64-bit)	GNU GCC- 4.5.2/ 4.7.2 / 4.8.3 / 5.2.0	share/PLI/VCS/LINUX64
SUSE (32-bit)	GNU GCC- 4.5.2/ 4.7.2 / 4.8.3 / 5.2.0	share/PLI/VCS/SUSE32
SUSE (64-bit)	GNU GCC- 4.5.2/ 4.7.2 / 4.8.3 / 5.2.0	share/PLI/VCS/SUSE64

SystemC Dumper for Synopsys VCS/VCS MX 2017.12 / SystemC 2.3.1

Platform	Compiler	Library Directory
Linux (32-bit)	GNU GCC- 4.7.2 / 4.8.3 / 5.2.0 / 6.2.0	share/PLI/VCS/LINUX
Linux (64-bit)	GNU GCC- 4.7.2 / 4.8.3 / 5.2.0 / 6.2.0	share/PLI/VCS/LINUX64
SUSE (32-bit)	GNU GCC- 4.7.2 / 4.8.3 / 5.2.0 / 6.2.0	share/PLI/VCS/SUSE32
SUSE (64-bit)	GNU GCC- 4.7.2 / 4.8.3 / 5.2.0 / 6.2.0	share/PLI/VCS/SUSE64

SystemC Dumper for Synopsys VCS/VCS MX 2018.09 / SystemC 2.2.0

Platform	Compiler	Library Directory
Linux (32-bit)	GNU GCC- 4.2.2 / 4.8.3 / 5.2.0	share/PLI/VCS/LINUX
Linux (64-bit)	GNU GCC- 4.2.2 / 4.8.3 / 5.2.0	share/PLI/VCS/LINUX64
SUSE (32-bit)	GNU GCC- 4.2.2 / 4.8.3 / 5.2.0	share/PLI/VCS/SUSE32
SUSE (64-bit)	GNU GCC- 4.2.2 / 4.8.3 / 5.2.0	share/PLI/VCS/SUSE64

SystemC Dumper for Synopsys VCS/VCS MX 2018.09 / SystemC 2.3.1

Platform	Compiler	Library Directory
Linux (32-bit)	GNU GCC- 4.8.3 / 5.2.0 / 6.2.0	share/PLI/VCS/LINUX
Linux (64-bit)	GNU GCC- 4.8.3 / 5.2.0 / 6.2.0	share/PLI/VCS/LINUX64
SUSE (32-bit)	GNU GCC- 4.8.3 / 5.2.0 / 6.2.0	share/PLI/VCS/SUSE32
SUSE (64-bit)	GNU GCC- 4.8.3 / 5.2.0 / 6.2.0	share/PLI/VCS/SUSE64

The libraries are located under the <*VERDI_HOME*>/ directory.

VCS simulator

Novas object files for FSDB dumping in mixed SystemC/Verilog designs are provided for the VCS simulator. To dump SystemC and Verilog signals to FSDB files, you can use the FSDB Verilog FSDB dumping commands at the VCS prompt or add the FSDB dumping commands to the Verilog code. The related Novas object files for FSDB dumping for mixed SystemC/Verilog designs with VCS are located under the `<VERDI_HOME>/share/PLI/VCS/${PLATFORM}`.

The environment variable, `LD_LIBRARY_PATH` should be set by pointing to the `<VERDI_HOME>/share/PLI/VCS/ ${PLATFORM}` directory before running the simulation, "simv".

NOTE: To specify FSDB dumping commands at the UCLI prompt, the `-load libnovas.so:FSDBDumpCmd` option is required when running the VCS compilation.

Linking the Novas Library with the VCS Simulator

NOTE: VCS2017.03 (LINUX) is used as an example in the following steps.

1. Set the system's shared library search path to include the directory for the Novas object files for FSDB dumping:

```
>setenv LD_LIBRARY_PATH \
<VERDI_HOME>/share/PLI/VCS/<PLATFORM>: \
$LD_LIBRARY_PATH
```

2. Use VCS to compile your design.

   Compile Design with `g++42`

   ```
   % vcs -cpp g++42 -sysc=220 *.v *.cpp
   +vcsd -debug_access +memcbk +v2k $OTHER_VCS_OPTIONS
   ```

   Compile Design with `g++52`

   ```
   % vcs -cpp g++52 -sysc=220 *.v *.cpp
   +vcsd -debug_access +memcbk +v2k $OTHER_VCS_OPTIONS
   ```

3. Run simulation.

   ```
   %./simv
   ```

Mixed SystemC/HDL Dumping Limitations

Non-intuitive names, such as `signal_1`, are shown as signal names if no signal names have been passed in a member initializer for the object.

VCS and Virtualizer

Verdi FSDB dumping in the mixed SystemC/Verilog designs supports the co-simulation of VCS and Virtualizer. To dump SystemC and Verilog signals to FSDB files, use the FSDB dumping commands at the VCS prompt or add the FSDB dumping commands to the Verilog code.

Linking the Novas Library with the VCS and Virtualizer:

1. Setup environment:

```
% setenv COWARE_CXX_COMPILER gcc-4.7.2-64
% setenv COWAREHOME $SNPS_VP_HOME
% source $SNPS_VP_HOME/setup.csh -pa
% setenv LD_LIBRARY_PATH ${VERDI_HOME}/share/PLI/VCS/
<PLATFORM>:$ LD_LIBRARY_PATH
% setenv FSDB_DUMP_RIDB 1
% setenv FSDB_ALL 1
```

2. Compile design:

```
% vlogan *.v
% vlogan *.v -sysc=snps_vp -sc_module *
% syscan -cflags -g *.cpp -sysc=snps_vp
% vcs -debug_access+r -sysc -sysc=snps_vp -lca sc_main -fsdb
```

3. Run simulation:

```
% ./simv -ucli
```

Linking with Cadence Simulators

NOTE: A *novas_dump.log* file gets created during simulation to record some essential simulation/dumping information, such as simulation options used, environment variable settings, and Novas object directories linked. This log file is needed to report a dumper related issue effectively.

IUS (13.1 or Later Versions) - Verilog Only

The Novas object file for FSDB dumping supports signal dumping in pure Verilog designs with IUS. The object file for pure Verilog designs with IUS is called `libpli.so` and is located under the `<VERDI_HOME>/share/PLI/IUS/${PLATFORM}` directory. To link this object file, set the `LD_LIBRARY_PATH` environment variable to point to the correct directory and then the correct file is loaded for FSDB Dumping.

Quick Start Steps

1. Specify the paths to the FSDB PLI library and dumper engine as paths of the `LD_LIBRARY_PATH` environment variable.

```
> setenv LD_LIBRARY_PATH \
<VERDI_HOME>/share/PLI/IUS/${PLATFORM}: \
$LD_LIBRARY_PATH
```

NOTE: This must be specified regardless of how the simulator is linked and run.

2. Specify FSDB dumping commands inside the design like:

```
top.v:

module top;
...
initial
  begin
```

```
    $fsdbDumpfile("my_nc_test.fsdb");
    $fsdbDumpvars;
    ..
end // initial block
```

3. Choose the preferred method to run IUS.

   Using "ncvlog", "ncelab", "ncsim" trilogy:

```
> ncvlog top.v
ncvlog top.v
ncvlog: 12.20-s017: (c) Copyright 1995-2013 Cadence Design
Systems, Inc.

> ncelab top -access +r
ncelab: 12.20-s017: (c) Copyright 1995-2013 Cadence Design
Systems, Inc.

> ncsim top
ncsim: 12.20-s017: (c) Copyright 1995-2013 Cadence Design
Systems, Inc.
*Verdi* Loading libsscore_ius122.so

ncsim> run
Simulation complete via $finish(1) at time 30 NS + 0
./top.v:13    #10 $finish;
ncsim> exit
```

   Using "ncverilog":

```
> ncverilog top.v -access +r
ncverilog: 12.20-s017: (c) Copyright 1995-2013 Cadence Design
Systems, Inc.

...
*Verdi* Loading libsscore_ius122.so
FSDB Dumper for IUS, Release Verdi_M-2017.03,
Linux, 10/20/2014 (C) 1996 - 2014 by Synopsys, Inc.
```

   Using "irun":

```
> irun top.v -access +r
irun: 12.20-s017: (c) Copyright 1995-2013 Cadence Design
Systems, Inc.

...
*Verdi* Loading libsscore_ius122.so
FSDB Dumper for IUS, Release Verdi_M-2017.03,
Linux, 10/20/2014 (C) 1996 - 2014 by Synopsys, Inc.
```

4. Use IUS with FSDB dumping commands specified in Tcl format, *cmd.tcl*.

   `cmd.tcl`:

```
call {$fsdbDumpfile} {"my.fsdb"}
call {$fsdbDumpvars} {top}
run
exit
```

Using "ncvlog", "ncelab", "ncsim" trilogy:

```
> ncvlog top.v
> ncelab top -access +r
> ncsim top -i cmd.tcl
```

Using "ncverilog":

```
> ncverilog top.v -access +r +tcl+cmd.tcl
```

Using "irun":

```
> irun top.v -access +r +tcl+cmd.tcl
```

5. Use IUS PLI boot method while FSDB dumping commands specified in Tcl format, *cmd.tcl*.

   Using "ncvlog", "ncelab", "ncsim" trilogy:

   ```
   > ncvlog top.v
   > ncelab top -access +r -loadpli1 debpli:novas_pli_boot
   > ncsim top -i cmd.tcl
   ```

   Using "ncverilog":

   ```
   > ncverilog top.v -access +r +tcl+cmd.tcl -loadpli1
   debpli:novas_pli_boot
   ```

   Using "irun":

   ```
   > irun top.v -access +r +tcl+cmd.tcl -loadpli1
   debpli:novas_pli_boot
   ```

Dynamic Linking

If the IUS execution files (*ncelab* and *ncsim*) being used do not statically link with any other applications, dynamic linking can be used to link the Novas shared library file (`libpli.so` or `libpli.sl`) for FSDB dumping with IUS for pure Verilog designs. These files can be found under the `<VERDI_HOME>/share/PLI/IUS/${PLATFORM}/` directories.

Specify Shared Library Search Path

IUS searches and loads only one *libpli.so* (*libpli.sl*) file from the shared library paths. The system shared library search path must be set before running *ncelab*, *ncsim* or *ncverilog*.

Solaris/Linux Platforms

```
> setenv LD_LIBRARY_PATH \
    ${VERDI_HOME}/share/PLI/IUS/${PLATFORM}/: \
    $LD_LIBRARY_PATH
```

Perform Elaboration and Run Simulation

If you use the *ncverilog* command to run IUS, you must specify the `-access +r (+ncaccess+r)` or `+linedebug` options to enable read access of the design object.

```
> ncverilog +ncaccess+r -f run.f
```

or

```
> ncverilog +linedebug -f run.f
```

You must specify `+linedebug` when running the `ncverilog` command if you want to set line break points or run the design step by step using interactive mode in the Verdi platform.

If you use the *ncvlog*, *ncelab* and *ncsim* trilogy to run the IUS simulation, you must specify the `-access +r` option with *ncelab* or specify the `-linedebug` option with *ncvlog* to enable read access of the design object.

```
> ncvlog -linedebug -f ncvlog.args
> ncelab -f ncelab.args
```

Or

```
> ncvlog -f ncvlog.args
> ncelab -access +r -f ncelab.args
```

Then,

```
> ncsim -f ncsim.args
```

You must specify `-linedebug` with `ncvlog` to compile the designs if you want to set line break points or run the design step by step using interactive mode in the Verdi platform.

You can check whether IUS linked with the Novas object file for FSDB dumping successfully by using the `call` command at the `ncsim` prompt (use `ncsim top -tcl` to enter `ncsim` prompt). Specifying the `call` command without any option lists all the Verilog FSDB dumping commands that are available in the IUS simulation. For example:

```
> ncsim -f ncsim.args -tcl
ncsim> call
Available C function names:
```

```
  fsdbDumpvars
  fsdbDumpvarsES
  fsdbDumpMDA
  fsdbDumpSVA
  fsdbDumpvarsByFile
  fsdbSuppress
  fsdbDumpon
  fsdbDumpoff
  fsdbSwitchDumpfile
  fsdbDumpfile
  fsdbAutoSwitchDumpfile
  fsdbDumpFinish
  fsdbDumpflush

$fsdbDumpvars
$fsdbDumpMDA
$fsdbDumpSVA
$fsdbDumpvarsByFile
$fsdbSuppress
$fsdbDumpon
$fsdbDumpoff
$fsdbSwitchDumpfile
$fsdbDumpfile
$fsdbAutoSwitchDumpfile
$fsdbDumpFinish
$fsdbDumpflush
$fsdbLog
```

Dynamic Loading

Dynamic loading is another easy way to link the Novas shared library file for FSDB dumping and other applications with IUS simultaneously. For dynamic loading with IUS, the Novas shared library file for FSDB dumping is `libIUS.so` These files can be found under the `<VERDI_HOME>/share/PLI/IUS/${PLATFORM}/` directories.

Specify Shared Library Search Path

Set the system shared library search path to include the directory for the Novas object file for FSDB dumping.

Solaris/Linux Platforms

```
> setenv LD_LIBRARY_PATH \
    ${VERDI_HOME}/share/PLI/IUS/${PLATFORM}/boot: \
    $LD_LIBRARY_PATH
```

Load and Link the Novas Object File

If you use the `ncverilog` command to run IUS, specify the `+loadpli1=debpli:novas_pli_boot` option with `ncverilog.debpli` as the library file name of the dynamic loading version of the FSDB dumper. `novas_pli_boot` is the booting function of the FSDB dumper. You must also specify the `+ncaccess+r` or `+linedebug` option to enable read access of the design object.

```
> ncverilog +ncaccess+r \
        +loadpli1=debpli:novas_pli_boot -f run.f
```

or

```
> ncverilog +linedebug \
        +loadpli1=debpli:novas_pli_boot -f run.f
```

NOTE: To load the CFC function library, add the additional option `-loadcfc debcfc:novas_cfc_boot` as follows:

```
> ncverilog +ncaccess+r \
 +loadpli1=debpli:novas_pli_boot \
 -loadcfc debcfc:novas_cfc_boot \
 -f run.f
```

or

```
> irun +ncaccess+r \
 +loadpli1=debpli:novas_pli_boot \
 -loadcfc debcfc:novas_cfc_boot \
 -f run.f
```

The message `*** Registering Novas CFC commands...` is displayed once it succeeds.

You must specify `+linedebug` with `ncverilog` if you want to set line break points or run the design step by step using interactive mode in the Verdi platform.

If you use the *ncvlog*, *ncelab*, and *ncsim* trilogy to run the IUS simulation, specify the `-loadpli1 debpli:novas_pli_boot` option with `ncelab`. You must specify the `-access +r` option with `ncelab` or specify the `-access +r` option with `ncvlog` to enable read access of the design object.

```
> ncvlog -linedebug -f ncvlog.args
> ncelab -loadpli1 debpli:novas_pli_boot -f ncelab.args
```

or

```
> ncvlog -f ncvlog.args
> ncelab -access +r -loadpli1 debpli:novas_pli_boot -f
ncelab.args
```

You must specify `-linedebug` with `ncvlog` to compile the design if you want to set line break points or run the design step by step using interactive mode in the Verdi platform.

NOTE: For backward compatibility, *deb_PLIPtr* or *debpli_boot* can be used as the booting function.

If IUS loads the FSDB dumper successfully, you will see the following message when *ncsim* starts,

```
*Verdi* Loading libsscore_ius122.so
```

Specifying the `call` command without any option lists all the Verilog FSDB dumping commands that are available in the IUS simulation. For example:

```
*Verdi* Loading libsscore_ius122.so
ncsim> run
Simulation stopped via $stop(1) at time 0 FS + 0
./a.v:8    $stop;
ncsim> call
Available C function names:
fsdbDumpvars
  fsdbDumpvarsES
  fsdbDumpMDA
  fsdbDumpSVA
  fsdbDumpvarsByFile
  fsdbSuppress
  fsdbDumpon
  fsdbDumpoff
  fsdbSwitchDumpfile
  fsdbDumpfile
  fsdbAutoSwitchDumpfile
  fsdbDumpFinish
  fsdbDumpflush

$fsdbDumpvars
$fsdbDumpMDA
$fsdbDumpSVA
$fsdbDumpvarsByFile
$fsdbSuppress
$fsdbDumpon
$fsdbDumpoff
$fsdbSwitchDumpfile
$fsdbDumpfile
$fsdbAutoSwitchDumpfile
$fsdbDumpFinish
$fsdbDumpflush
$fsdbLog
```

IUS (IUS13.1 and Later Versions) - Verilog, VHDL or Mixed

The Novas object file for FSDB dumping in the `${VERDI_HOME}/share/PLI/IUS/${PLATFORM}` directory is for both a VPI-based and VHPI-based FSDB dumping and supports signal dumping in pure VHDL or mixed-HDL designs.

There are several files included in the directory.

`libpli.so`	`${VERDI_HOME}/share/PLI/IUS/${PLATFORM}/`	FSDB dumping task library
`libcfc.so`	`${VERDI_HOME}/share/PLI/IUS/${PLATFORM}/`	C command library
`libfmi.so`	`${VERDI_HOME}/share/PLI/IUS/${PLATFORM}/`	VHDL procedure library
`novas.vhd`	`${VERDI_HOME}/share/PLI/IUS/${PLATFORM}/`	VHDL procedure
`libsscore_iusXXX.so`	`${VERDI_HOME}/share/PLI/lib/${PLATFORM}/`	FSDB dumper engine library

The `libpli.so` file provides the FSDB Verilog FSDB dumping commands. The `libfmi.so` file provides the FSDB VHDL foreign functions, and the `libcfc.so` file provides the FSDB C commands. The `novas.vhd` file contains the FSDB dumping commands in VHDL procedure format. The `libsscore_iusXXX.so` file is the FSDB dumper engine library.

If you use any of the three previously mentioned programming interfaces as provided by other vendors or your own, you may have to merge and rebuild these shared libraries. Otherwise, you can link the FSDB dumper with the *Dynamic Loading* (bootstrap) method. The detailed steps for merging these files with other vendors are discussed later.

Dynamic Linking

By default, IUS automatically searches for and loads the `libpli.so`, `libfmi.so`, and `libcfc.so` files according to the `LD_LIBRARY_PATH` setting. If you do not use any PLI, CFC or FMI programs from other vendors or your own, you can directly use these files from the Novas package.

To link the Novas object files for FSDB dumping with IUS, specify the shared library path, and then compile and include the FSDB dumping commands.

Specify Shared Library Search Path

Set the system shared library search path to include the directory of the Novas object file for FSDB dumping before running *ncelab* and *ncsim*.

Solaris/Linux Platforms

```
> setenv LD_LIBRARY_PATH \
  ${VERDI_HOME}/share/PLI/IUS/${PLATFORM}: \
  $LD_LIBRARY_PATH
```

Compile and Include FSDB Dumping Commands

When invoking FSDB dumping commands at the simulator command prompt ("ncsimv >") or in VHDL design is needed, one of the following methods must be used to compile and include the FSDB dumping commands.

Method 1: Using ncvhdl, ncvlog, ncelab and ncsim:

1. Compile the *novas.vhd* file into the default design working library. This VHDL file is found in the `<VERDI_HOME>/share/PLI/IUS/${PLATFORM}` directory.

   ```
   > ncvhdl -message \
     ${VERDI_HOME}/share/PLI/IUS/${PLATFORM}/novas.vhd
   ```

   The `novas.vhd` file contains the declarations of the FSDB foreign functions in the *pkg* and *novas* packages.

2. Use the novas package in any VHDL files that call any FSDB foreign functions. Assume the `novas.vhd` file is compiled into the default working library. Use the *novas* package in the VHDL code as shown in the following example:

   ```
   library IEEE;
   use IEEE.std_logic_1164.all;
   library work;
   use work.novas.all;
   ```

   The following example shows how to include the FSDB foreign functions in the VHDL code.

   ```
   process
   begin
      fsdbDumpfile("test.fsdb");
      fsdbDumpvars(0, ":");
      wait ;
   end process;
   ```

Re-compile any VHDL files that were modified. To learn more about the usage and syntax of the FSDB foreign functions, refer to the *FSDB Dumping Commands Used in VHDL* section.

Enable Read Access

To enable read access of the design objects, the `-linedebug` option must be included with *ncvhdl*, *ncvlog*, or the `-access +r` option specified with *ncelab*. For example:

```
> ncvlog -message -linedebug test.v
```
or
```
> ncvhdl -message -linedebug test.vhd
```
or
```
> ncelab -message -access +r WORK.TOP:BLK
```

The `-linedebug` command line option must be specified with *ncvhdl* or *ncvlog* to compile the design if you want to set line break points or run the design step by step using interactive mode in the Verdi platform.

Method 2: Using irun:

Use the novas package in any VHDL files that call any FSDB foreign functions. Assume the `novas.vhd` file is compiled into the default working library. Use the *novas* package in the VHDL code as shown in the following example:

```
library IEEE;
use IEEE.std_logic_1164.all;
library work;
use work.novas.all;
```

The following example shows how to include the FSDB foreign functions in the VHDL code.

```
process
begin
   fsdbDumpfile("test.fsdb");
   fsdbDumpvars(0, ":");
   wait ;
end process;
```

Re-compile any VHDL files that were modified. To learn more about the usage and syntax of the FSDB foreign functions, refer to the *FSDB Dumping Commands Used in VHDL* section.

NOTE: The `novas.vhd` file must be compiled before other files which use `use work.novas.all`. For example, assume the design file `vhdl_design.f` contains file paths for

VHDL design portion:
```
./src/vhdl/TOP.vhd
./src/vhdl/sytem.vhd
./src/vhdl/tb_CPUsystem.vhd
...
```
Then the path of `novas.vhd` must be listed before other files which include `use work.novas.all.`
```
${VERDI_HOME}/share/PLI/IUS/LINUX/novas.vhd
./src/vhdl/TOP.vhd
./src/vhdl/sytem.vhd
./src/vhdl/tb_CPUsystem.vhd
...
```

Enable Read Access

To enable read access of the design objects, the `-linedebug` option must be included. For example:

```
> irun -message -linedebug –f vhdl_design.f –f verilog_design.f
–loadpli1 debpli:novas_pli_boot
```

When the linking is successful, a Verdi banner is shown to indicate the coordinating FSDB dumper engine is loading:

```
*Novas* Loading libsscore_iusXXX.so
```

Dynamic Loading

Additional options can be specified to tell IUS to load the PLI/CFC/FMI libraries for the Novas FSDB dumper from a non-default filename.

`debpli.so`	`${VERDI_HOME}/share/PLI/IUS/${PLATFORM}/boot`	FSDB Dumper task library
`debcfc.so`	`${VERDI_HOME}/share/PLI/IUS/${PLATFORM}/boot`	C command library
`debfmi.so`	`${VERDI_HOME}/share/PLI/IUS/${PLATFORM}/boot`	VHDL procedure library
`novas.vhd`	`${VERDI_HOME}/share/PLI/IUS/${PLATFORM}/boot`	VHDL procedure

It is convenient to directly use the FSDB dumper without rebuilding new libraries to merge with other third party programs. To dynamically load the FSDB dumper with IUS, specify the shared library path, and then compile and include the FSDB dumping commands.

Specify Shared Library Search Path

Set the system shared library search path to include the directory of the dynamic loading version of the FSDB dumper before running *ncelab* and *ncsim*.

Solaris/Linux Platforms

```
> setenv LD_LIBRARY_PATH \
  ${VERDI_HOME}/share/PLI/IUS/${PLATFORM}/boot:\
  $LD_LIBRARY_PATH
```

Compile and Include FSDB Dumping Commands

This is optional. However, if the FSDB dumping commands are invoked at the simulator prompt in the VHDL design or the simulation is run using interactive mode in the Verdi platform, the following method must be used to compile and include the FSDB dumping commands:

Method: Using ncvhdl, ncvlog, ncelab and ncsim:

1. Compile the `novas.vhd` file into the default design working library. You can find this VHDL file in the `<VERDI_HOME>/share/PLI/IUS/${PLATFORM}` directory.

   ```
   > ncvhdl -message \
     ${VERDI_HOME}/share/PLI/IUS/${PLATFORM}/novas.vhd
   ```

   The `novas.vhd` file contains the declarations of the FSDB foreign functions in the *pkg* and *novas* packages.

2. Use the novas package in any VHDL files that call any FSDB foreign functions. Assume the `novas.vhd` file is compiled with the default working library. Use the *novas* package in the VHDL code as shown in the following example:

   ```
   library IEEE;
   use IEEE.std_logic_1164.all;
   library work;
   use work.novas.all;
   ```

   The following example shows how to include the FSDB foreign functions in the VHDL code:

   ```
   process
   begin
      fsdbDumpfile("test.fsdb");
      fsdbDumpvars(0, ":");
      wait ;
   end process;
   ```

Recompile any VHDL files that were modified. To learn more about the usage and syntax of the FSDB foreign functions, refer to the *FSDB Dumping Commands Used in VHDL* section.

3. To enable read access of the design objects, the `-linedebug` option must be included with *ncvhdl*, *ncvlog*, or the `-access +r` option specified with *ncelab*. For example:

```
> ncvlog -message -linedebug test.v
```

or

```
> ncvhdl -message -linedebug test.vhd
```

or

```
> ncelab -message -access +r WORK.TOP:BLK
```

This is optional. However, if you want to set line break points or run the design step by step using interactive mode in the Verdi platform, the `-linedebug` command line option must be specified with *ncvlog* to compile the design.

```
> ncvlog -message -linedebug test.v
```

or

```
> ncvhdl -message -linedebug test.vhd
```

4. Specify the `-loadpli1 debpli:novas_pli_boot` option with *ncelab*. The `-access +r` option must also be specified with ncelab to enable read access of the design object.

```
> ncelab -access +r -loadpli1 debpli:novas_pli_boot \
         -VHDLEXPAND WORK.TOP:BLK
```

NOTE: From Cadence IUS 5.5 and later versions, the symbols in PLI applications are not exported by default. This may cause a crash if you load several PLI applications that contain the FSDB writer library. To avoid the crash, you can add the `:export` qualifier to each `-loadpli` option. This exports the symbols in the PLI applications. For example:

```
>ncelab -access +r \
-loadpli1 debpli:novas_pli_boot:export\
-VHDLEXPAND WORK.TOP:BLK
```

Instead of adding the `:export` qualifier, you can use the `-pli_export` option. For example:

```
>ncelab -access +r \
-loadpli1 debpli:novas_pli_boot \
-VHDLEXPAND WORK.TOP:BLK -pli_export
```

---

5. Specify the `-loadcfc debcfc:novas_cfc_boot` and `-loadfmi debfmi:novas_fmi_boot` options with *ncsim*.

```
> ncsim -loadcfc debcfc:novas_cfc_boot \
     -loadfmi debfmi:novas_fmi_boot \
     WORK.TOP:BLK -tcl
```

NOTE: For backward compatibility, *debcfc_boot* can be used as the booting function of *debcfc* and use *debfmi_boot* as the booting function of *debfmi*.

When the linking is successful, a Novas banner is shown to indicate the coordinating FSDB dumper engine is loading:

```
*Verdi* Loading libsscore_ius122.so
```

NOTE: To load the CFC function library with irun, add the additional option `-loadcfc debcfc:novas_cfc_boot` as follows:

```
> irun +ncaccess+r \
   +loadpli1=debpli:novas_pli_boot \
   -loadcfc debcfc:novas_cfc_boot \
   <file(s)>
```

The message `*** Registering Novas CFC commands...` is displayed once it succeeds.

Calling CFC Functions from the ncsim Prompt

To check whether IUS linked the Novas object file for FSDB dumping successfully, use the `call` command at the *ncsim* prompt. Specifying the `call` command without any options will list all CFC commands and Verilog FSDB dumping commands that are available in *ncsim*. For example:

```
ncsim> call

*** Registering Novas CFC commands...
Available C function names:
  fsdbDumpvars
  fsdbDumpvarsES
  fsdbDumpMDA
  fsdbDumpSVA
  fsdbDumpvarsByFile
  fsdbSuppress
  fsdbDumpon
  fsdbDumpoff
  fsdbSwitchDumpfile
  fsdbDumpfile
  fsdbAutoSwitchDumpfile
  fsdbDumpFinish
  fsdbDumpflush
```

```
$fsdbDumpvars
$fsdbDumpMDA
$fsdbDumpSVA
$fsdbDumpvarsByFile
$fsdbSuppress
$fsdbDumpon
$fsdbDumpoff
$fsdbSwitchDumpfile
$fsdbDumpfile
$fsdbAutoSwitchDumpfile
$fsdbDumpFinish
$fsdbDumpflush
$fsdbLog
```

The FSDB dumping commands can be called at the *ncsim* prompt to dump the design directly. In the following example, two FSDB dumping commands are called at the *ncsim* prompt to dump the entire design to *test.fsdb*:

```
ncsim> call fsdbDumpfile test.fsdb
ncsim> call fsdbDumpvars 0 :
```

For more usage information about these FSDB commands, refer to the *FSDB Dumping Commands Used in VHDL* section.

NOTE: If you run a pure Verilog design or mixed-HDL design with a Verilog top module and Novas VHPI/VPI, follow the IUS signals/scopes naming convention for pure Verilog design to call Novas VHPI/VPI FSDB dumping commands. If you run a pure VHDL design or mixed-HDL design with VHDL top module and Novas VHPI/VPI, follow the NC-SIM signals/scopes naming convention for pure VHDL design to call Novas VHPI/VPI FSDB dumping commands.

IUS (IUS13.1 and Later Versions) - SystemC and HDL (Verilog/SystemVerilog, VHDL) Mixed

SystemC Dumper for Cadence Simulator

Simulator Version	Platform	Compiler	Library Directory
IUS 13.1/13.2 14.1/14.2 15.1	Sun Solaris 5.10 (32-bit)	GNU GCC-4.1	No Support
	Linux (32-bit)	GNU GCC-4.1	share/PLI/IUS/LINUX
	Linux (64-bit)	GNU GCC-4.1	share/PLI/IUS/LINUX64
	SUSE (32-bit)	GNU GCC-4.1	share/PLI/IUS/SUSE32
	SUSE (64-bit)	GNU GCC-4.1	share/PLI/IUS/SUSE64
	Sun Solaris 5.10 (32-bit)	GNU GCC-4.4	No Support
	Linux (32-bit)	GNU GCC-4.4	share/PLI/IUS/LINUX
	Linux (64-bit)	GNU GCC-4.4	share/PLI/IUS/LINUX64
	SUSE (32-bit)	GNU GCC-4.4	share/PLI/IUS/SUSE32
	SUSE (64-bit)	GNU GCC-4.4	share/PLI/IUS/SUSE64

The libraries are located under the `<VERDI_HOME>/` directory.

NC-SystemC Simulator

The following functions are provided to create an FSDB file that contains the values of signals when values of signals change during the simulation:

Functions

- fsdbDumpfile
- fsdbDumpSC
- fsdbSwitchDumpfile
- fsdbAutoSwitchDumpfile
- fsdbDumpFinish
- fsdbDumpon
- fsdbDumpoff

Linking the FSDB Dumper Library with the NC-SystemC Simulator

NOTE: Linux and IUS13.1 are used as an example in the following steps. These steps are valid only when the dumping commands are specified inside the SystemC source code (*a .cpp* file).

To link the FSDB Dumper Library with the NC-SystemC simulator, use the following steps:

1. Include the header file *fsdb_nc_mix.h.*

```
#include "systemc.h"
#include "fsdb_nc_mix.h"
...
```

2. Invoke the dumping functions to dump module instance(s) or signal(s) explicitly.

```
fsdbDumpSC(0,"my_system");
```

3. Set the library path.

```
>setenv LD_LIBRARY_PATH
<VERDI_HOME>/share/PLI/IUS/<PLATFORM>: \
$LD_LIBRARY_PATH
```

Simulate SystemC Design with ncsc_run

4. Compile your SystemC Design with ncsc_run and run the simulation.

```
>ncsc_run *.cpp *.v *.vhdl -TOP <top module>
```

   To dump the signals in your mixed SystemC/HDL design, modify your SystemC code with the following steps:

```
-I<VERDI_HOME>/share/PLI/IUS/<PLATFORM> \
-L<VERDI_HOME>/share/PLI/IUS/<PLATFORM> \
-lfsdbSC
```

Mixed SystemC/HDL Dumping Limitations

Non-intuitive names, such as, *signal_1*, are shown as signal names if no signal names are passed in a member initializer for the object.

SystemC Dumper for Cadence SCV Transaction

To dump transactions, your SystemC code must be modified with the following steps:

1. Include the header file *scv_tr_fsdb.h*.

   ```
   #include "scv.h"
   #include "scv_tr_fsdb.h"
   ```

   Include `-I<VERDI_HOME>/share/PLI/IUS/LINUX` in the c++ compiler option you specify to compile your design.

2. Invoke `scv_tr_fsdb_init()` before opening a file for recording the transactions.

   ```
   int sc_main (int argc , char *argv[]) {
   scv_tr_fsdb_init();
   scv_tr_db db("my_db");
   scv_tr_db::set_default_db(&db);
   ...
   }
   ```

Then the transaction which records the classes and methods you invoked in your design is saved in an FSDB file.

If you use the SCV (CVE) library of Cadence **NCSC13.1**, **NCSC13.2**, **NCSC14.1**, **NCSC14.2**, and **NCSC15.1**, the current release supports the following libraries for the various C++ compilers and platforms:

SystemC Verification Transaction Recording Library for Cadence

Simulator Version	Platform	Compiler	Library Directory
IUS 13.1/13.2 14.1/14.2 15.1	Sun Solaris 5.10 (32-bit)	GNU GCC-4.1	No Support
	Linux (32-bit)	GNU GCC-4.1	share/PLI/IUS/LINUX
	Linux (64-bit)	GNU GCC-4.1	share/PLI/IUS/LINUX64
	SUSE (32-bit)	GNU GCC-4.1	share/PLI/IUS/SUSE32
	SUSE (64-bit)	GNU GCC-4.1	share/PLI/IUS/SUSE64
	Sun Solaris 5.10 (32-bit)	GNU GCC-4.4	No Support
	Linux (32-bit)	GNU GCC-4.4	share/PLI/IUS/LINUX
	Linux (64-bit)	GNU GCC-4.4	share/PLI/IUS/LINUX64
	SUSE (32-bit)	GNU GCC-4.4	share/PLI/IUS/SUSE32
	SUSE (64-bit)	GNU GCC-4.4	share/PLI/IUS/SUSE64

The libraries are located under the `<VERDI_HOME>/` directory.

To dynamically link the simulation execution file with the SCV FSDB transaction recording library, you must set the system environment variable `LD_LIBRARY_PATH` to include the directories of `libscv_tr_fsdb.so`.

```
> setenv LD_LIBRARY_PATH
${VERDI_HOME}/share/PLI/IUS/${PLATFORM}:
$LD_LIBRARY_PATH
```

If you use the SCV (CVE) library of Cadence IUS 13.1 and use gcc 4.1.x on Linux (64bit for example), you need to specify additional options to let NCSC dynamically link the SCV FSDB transaction recording library and the FSDB writer into the final simulation file. For example:

```
> cve main.cpp mybus_data.cpp mybus_master.cpp -DSDI -GNU
-L${VERDI_HOME}/share/PLI/IUS/LINUX64
-lscv_tr_fsdb
```

Linking with ModelSim Simulators

NOTE: A `novas_dump.log` file gets created during simulation to record some essential simulation/dumping information, such as simulation options used, environment variable settings, and Novas object directories linked. This log file is needed to report dumper related issue effectively.

ModelSim (10.2 or Later Versions) - Verilog Only

The Novas object files for FSDB dumping support signal dumping in pure Verilog designs with ModelSim. To dump Verilog signals to the FSDB files, you can use the FSDB dumping commands at the `vsim` prompt or in the Verilog code.

The related files for FSDB dumping of pure Verilog designs with ModelSim are in the `<VERDI_HOME>/share/PLI/MODELSIM/${PLATFORM}` directory. The FSDB dumper is a shared library file called `novas_fli.so`. It must be loaded in *vsim* when running the ModelSim simulation.

To link the Novas object files for FSDB dumping with ModelSim, specify the shared library path and then load the FSDB dumper.

Specify Shared Library Search Path

Set the system shared library search path to include the directory of the Novas object files for FSDB dumping.

Solaris/Linux Platforms

```
> setenv LD_LIBRARY_PATH \
  ${VERDI_HOME}/share/PLI/MODELSIM/${PLATFORM}:\
  $LD_LIBRARY_PATH
```

Load the FSDB Dumper

There are three methods to load the FSDB dumper `novas_fli.so` into *vsim* during simulation.

Method 1: -pli Option

Use the `-pli` option with *vsim*. For example:

Solaris/Linux Platforms

```
> vsim -pli novas_fli.so top ...
```

Method 2: veriuser Entry

Set the *veriuser* entry in the `modelsim.ini` file. For example:

Solaris/Linux Platforms

```
[vsim]
...
veriuser=novas_fli.so
```

Method 3: PLIOBJS Environment Variable

Set the *PLIOBJS* system environment variable before running *vsim*. For example:

Solaris/Linux Platforms

```
> setenv PLIOBJS "novas_fli.so"
```

When the linking is successful, a Novas banner is shown to indicate the coordinating FSDB dumper engine is loading; otherwise, a failed to load message is shown instead.

Succeeds:

```
*Novas* Loading libssocre_mtiXXX.so
```

Fails:

```
*Novas* Failed to load FSDB dumper.
```

Linking with Other Applications

Other vendors often provide their applications as standalone shared library files. One of the following methods can be used to load different PLI programs into *vsim* at the same time.

Method 1: -pli Option

Use the `-pli` option with *vsim*. For example:

Solaris/Linux Platforms

```
> vsim -pli novas_fli.so -pli third_party.so top ...
```

Method 2: veriuser Entry

Set the *veriuser* entry in the `modelsim.ini` file. For example:

Solaris/Linux Platforms

```
[vsim]
...
veriuser=novas_fli.so third_party.so
```

Method 3: PLIOBJS Environment Variable

Set the *PLIOBJS* system environment variable before running *vsim*. For example:

Solaris/Linux Platforms

```
> setenv PLIOBJS "novas_fli.so third_party.so"
```

ModelSim (SE 10.2 or Later Versions) - VHDL or Mixed Only

The Novas object files for FSDB dumping support signal dumping in pure VHDL or mixed-HDL designs with ModelSim. To dump VHDL and Verilog signals to the FSDB files, you can use the FSDB dumping commands at the *vsim* prompt, invoke the FSDB foreign functions in the VHDL code or invoke the FSDB dumping commands in the Verilog code.

The related files for FSDB dumping of VHDL or mixed-HDL designs with ModelSim are in the `<VERDI_HOME>/share/PLI/MODELSIM/${PLATFORM}` directory. The FSDB dumper is a shared library file called `novas_fli.so`. You have to load it in *vsim* when running the ModelSim simulation.

NOTE: Limitation: The FSDB dumper for ModelSim cannot automatically record each value change of variable types. You must use *fsdbDumpVariable* to repeatedly dump the current value of a variable.

To link the Novas object files for FSDB dumping with ModelSim, specify the shared library path, compile and include the FSDB dumping commands, and then load the FSDB dumper.

Specify Shared Library Search Path

Set the system shared library search path to include the directory for the Novas object files for FSDB dumping.

Solaris/Linux Platforms

```
> setenv LD_LIBRARY_PATH \
  ${VERDI_HOME}/share/PLI/MODELSIM/${PLATFORM}:\
  $LD_LIBRARY_PATH
```

Compile and Include Novas FSDB Dumping Commands

Compile Novas FSDB Dumping Commands to Design Library

Compile the VHDL file `novas.vhd` into the default design working library or the *novas* design library. This VHDL file is found in the `<VERDI_HOME>/share/PLI/MODELSIM/${PLATFORM}` directory.

For example, compile `novas.vhd` into the specific design library:

```
> setenv MTI_VCO_MODE 32 // Enable 32-bit simulation
```

or

```
> setenv MTI_VCO_MODE 64 // Enable 64-bit simulation

> vlib novas
> vcom -work novas \
  ${VERDI_HOME}/share/PLI/MODELSIM/${PLATFORM}/novas.vhd
```

Or, compile `novas.vhd` into the default working design library:

```
> vcom ${VERDI_HOME}/share/PLI/MODELSIM/${PLATFORM}/novas.vhd
```

The `novas.vhd` file contains the declarations of the FSDB foreign functions in the *pkg* package and the definition of the *novas* entity.

Include Novas FSDB Dumping Commands in Design Files

Alternatively, use the *pkg* package in any VHDL design files that include FSDB foreign functions. For example, if the `novas.vhd` is compiled into the *novas* library:

```
library IEEE;
use IEEE.std_logic_1164.all;
library novas;
use novas.pkg.all;
```

Otherwise, if the *novas.vhd* is compiled into the default working library:

```
library IEEE;
use IEEE.std_logic_1164.all;
library work;
use work.pkg.all;
```

The following example shows how to invoke the FSDB foreign functions in the VHDL code.

```
process
begin
   fsdbDumpfile("test.fsdb");
   fsdbDumpvars(0, "system");
   wait ;
end process;
```

Re-compile all VHDL files you modified.

To learn more about the usage and syntax of the FSDB foreign functions, you can refer to the *FSDB Dumping Commands Used in VHDL* section.

Load the FSDB Dumper

The `novas_fli.so` Novas object files for FSDB dumping also support the FSDB dumping commands for Verilog. If any FSDB dumping commands are included in the Verilog code, *vsim* must load the `novas_fli.so` file as a PLI program. This is optional. There are three methods for loading the FSDB dumper:

Method 1: -pli Option

Use the `-pli` option with *vsim*. For example:

Solaris/Linux Platforms

```
> vsim -pli novas_fli.so novas top ...
```

Method 2: veriuser Entry

Set the *veriuser* entry in the `modelsim.ini` file. For example:

Solaris/Linux Platforms

```
[vsim]
...
veriuser= novas_fli.so
```

Method 3: PLIOBJS Environment Variable

Set the *PLIOBJS* system environment variable before running *vsim*. For example:

Solaris/Linux Platforms

```
> setenv PLIOBJS "novas_fli.so"
```

Loading the FSDB dumper is optional. However, if FSDB commands are invoked at the *vsim* prompt, or if the ModelSim simulation is run using interactive mode in the Verdi platform, the *novas* entity must be loaded when starting the ModelSim simulation with *vsim*. For example, assume `novas.vhd` is compiled into the *novas* library:

Solaris/Linux Platforms

```
> vsim novas.novas top ...
```

Then the FSDB commands can be invoked at the *vsim* prompt. For example:

```
VSIM 1> fsdbDumpfile test.fsdb (Creates the FSDB file)
VSIM 2> fsdbDumpvars 0 system  (Dump all signals from the system
region)
```

The `-sva` and `-assertdebug` options should be specified in the vsim command line options if dumping assertions is desired. For example,

```
> vsim -c top –pli novas_fli.so -sva -assertdebug
```

To learn more about the usage and syntax of the FSDB foreign functions, you can refer to the *FSDB Dumping Commands Used in VHDL* section.

ModelSim (SE 10.2 and Later Versions) - SystemC and HDL (Verilog/ SystemVerilog, VHDL) Mixed

SystemC Dumper for ModelSim 10.2

Platform	Compiler	Library Directory
Sun Solaris (32-bit)	Latest simulator supported compiler	share/PLI/MODELSIM/SOL2
Sun Solaris (64-bit)	Latest simulator supported compiler	share/PLI/MODELSIM/SOL7
Linux (32-bit)	Latest simulator supported compiler	share/PLI/MODELSIM /LINUX
Linux (64-bit)	Latest simulator supported compiler	share/PLI/MODELSIM / LINUX64
SUSE (32-bit)	Latest simulator supported compiler	share/PLI/MODELSIM /SUSE32
SUSE (64-bit)	Latest simulator supported compiler	share/PLI/MODELSIM /SUSE64

The libraries are located under the `<VERDI_HOME>/` directory.

ModelSim Simulator

The Novas object files for FSDB dumping support signal dumping in mixed SystemC/HDL designs with ModelSim. To dump signals to the FSDB files, there are two ways: 1) use the FSDB commands at the `vsim` prompt and include FSDB foreign functions in the VHDL code or 2) include the FSDB dumping commands in the Verilog code. The related Novas object files for FSDB dumping of mixed-SystemC/HDL designs with ModelSim are in the `<VERDI_HOME>/share/PLI/MODELSIM/${PLATFORM}` directory. The FSDB dumper is a shared library file called `novas_fli.so`. You must load it in vsim when running the ModelSim simulation.

Linking the Novas Library with the ModelSim Simulator

NOTE: Linux and Modelsim10.2 are used as examples in the following steps.

1. Set the shared library search path of the system to include the directory for the Novas object files for FSDB dumping.

```
>setenv LD_LIBRARY_PATH
<VERDI_HOME>/share/PLI/MODELSIM/${PLATFORM}:
$LD_LIBRARY_PATH
```

Compile Design

2. Create the library.

```
>vlib novas
```

3. Do common elaboration on Verilog design.

```
>vlog +define+NEW_PLI=1 -sv *.v
```

4. Compile the *.cpp codes.

```
>sccom -g *.cpp
Link the *.o files to be "systemc.so".
>sccom -link
```

5. Load the Novas object files for FSDB dumping while running the simulation:

```
>vsim -pli \
<VERDI_HOME>/share/PLI/MODELSIM/<PLATFORM>/novas_fli.so \
sc_main -c -do "fsdbDumpfile sc.fsdb; fsdbDumpSC 0; run 1000
ns; exit"
```

   Also, you can invoke the FSDB commands at the `vsim` prompt. For example:

```
VSIM 1>fsdbDumpfile test.fsdb
VSIM 2>fsdbDumpSC 0 system (Dump all signals from the system
region)
```

Mixed SystemC/HDL Dumping Limitations

Non-intuitive names, such as, *signal_1*, are shown as signal names if no signal names are passed in a member initializer for the object.

SystemC Dumper to Open SystemC Initiative Simulator

Simulators, Platforms, and Compilers Supported by Verdi

Automatic SystemC dumper for Open SystemC Inituative 2.0.1

Platform	Compiler	Library Directory
Sun Solaris 2.8 (32-bit)	GNU GCC-3.2	systemc/old/osci201/lib-solaris_gcc3_2
Linux Redhat 7.2 (32-bit)	GNU GCC-3.2	systemc/old/osci201/lib-linux_gcc3_2

Automatic SystemC Dumper for Open SystemC Initiative 2.1.v1

Platform	Compiler	Library Directory
Sun Solaris 2.8 (32-bit)	GNU GCC-3.2	systemc/old/osci21v1/lib-solaris_gcc3_2
Linux Redhat 7.2 (32-bit)	GNU GCC-3.2	systemc/old/osci21v1/lib-linux_gcc3_2
	GNU GCC-3.3.2	systemc/old/osci21v1/lib-linux_gcc3_32

SystemC Dumper for Open SystemC Initiative 2.0.1

Platform	Compiler	Library Directory
Sun Solaris 2.7/2.8 (32-bit)	GNU GCC-2.95	systemc/old/osci201/lib-solaris_gcc2_95
	GNU GCC-3.2	systemc/old/osci201/lib-solaris_gcc3_2

Linux Redhat 7.2 (32-bit)	GNU GCC-2.96	systemc/old/osci201/lib-linux_gcc2_96
	GNU GCC-3.2	systemc/old/osci201/lib-linux_gcc3_2
Linux Redhat 6.2 (32-bit)	GNU GCC-2.95	systemc/old/osci201/lib-linux_gcc2_95
x86/MS-Windows NT	MS Visual C++ 6.0	systemc/old/osci201/msvc60

SystemC Dumper for Open SystemC Initiative 2.1

Platform	Compiler	Library Directory
Sun Solaris 2.7/2.8 (32-bit)	GNU GCC-2.95	systemc/old/osci21/lib-solaris_gcc2_95
	GNU GCC-3.2	systemc/old/osci21/lib-solaris_gcc3_2
Linux Redhat 7.2 (32-bit)	GNU GCC-2.96	systemc/old/osci21/lib-linux_gcc2_96
	GNU GCC-3.2.3	systemc/old/osci21/lib-linux_gcc3_23
Linux Redhat 6.2 (32-bit)	GNU GCC-2.95	systemc/old/osci21/lib-linux_gcc2_95

SystemC Dumper for Open SystemC Initiative 2.1v1

Platform	Compiler	Library Directory
Sun Solaris 2.7/2.8 (32-bit)	GNU GCC-2.95	systemc/old/osci21v1/lib-solaris_gcc2_95
	GNU GCC-3.2	systemc/old/osci21v1/lib-solaris_gcc3_2
Linux Redhat 7.2 (32-bit)	GNU GCC-2.96	systemc/old/osci21v1/lib-linux_gcc2_96
	GNU GCC-3.2.3	systemc/old/osci21v1/lib-linux_gcc3_23
	GNU GCC-2.3.2	systemc/old/osci21v1/lib-linux_gcc3_2

SystemC Dumper for Open SystemC Initiative 2.2

Platform	Compiler	Library Directory
Linux Redhat 7.2 (32-bit)	GNU GCC-3.2.3	systemc/old/osci22/lib-linux_gcc3_23
	GNU GCC-3.4.x	systemc/old/osci22/lib-linux_gcc3_4x
	GNU GCC-4.0.2	systemc/old/osci22/lib-linux_gcc4_02
	GNU GCC-4.1.1	systemc/old/osci22/lib-linux_gcc4_11
Linux Redhat 7.2 (64-bit)	GNU GCC-4.0.2	systemc/old/osci22/lib-linux_gcc4_02_x86_64
	GNU GCC-4.1.1	systemc/old/osci22/lib-linux_gcc4_11_x86_64

The libraries are located under the *<VERDI_HOME>/share/PLI* directory.

Using the Automatic SystemC Dumper to Dump SystemC Designs

The *libfsdbSC.a* library provides the dumping functions that let you create an FSDB file containing the values of signals as they change during the simulation. Access to the dumping functions are provided in the header file named *fsdb_trace_file.h*.

The following functions are provided:

Functions

- fsdbDumpfile
- fsdbDumpAll
- fsdbDumpvars
- fsdbDumpAllTrans
- fsdbDumpTrans
- fsdbDumpon
- fsdbDumpoff
- fsdbEnableDumpVariable
- fsdbDumpFinish

- fsdbSwitchDumpfile
- fsdbAutoSwitchDumpfile

To dump the signals in your SystemC design, you must modify your SystemC code with the following steps:

1. Include the header file *fsdb_trace_file.h*.

```
#include "systemc.h"
#include "fsdb_trace_file.h"
...
```

2. Invoke the dumping functions to dump module instances or signals explicitly.

```
fsdbDumpvars(5,top);
```

3. Link the *libnffw.a* and *libfsdbSC.a* libraries.

   You can find libnffw.a in the *<VERDI_HOME>/share/FsdbWriter/$PLATFORM* sub-directory.

Linking the Novas Library with the Open SystemC Initiative Simulator

NOTE: The osci21v1 simulator in the Linux is used as an example in the following steps.

To build a SystemC executable that contains the Novas PLI application functions, use the following steps:

1. Set the library path.

```
>setenv LD_LIBRARY_PATH
<VERDI_HOME>/share/FsdbWriter/LINUX_GNC_32:\
$LD_LIBRARY_PATH
```

2. Compile your design. You should specify the compiler option *-gdwarf-2* and *-I<VERDI_HOME>/share/PLI/systemc/old/osci21v1/include*.

```
>g++ -gdwarf-2 -c *.cpp
-I<OSCI_INSTALL_DIR>/include
-I<VERDI_HOME>/share/PLI/systemc/osci21v1/include
```

3. Build an executable file. You should link the *libfsdbSC.a* and *libnffw.a* libraries.

```
>g++ *.o -o run.x
-L<VERDI_HOME>/share/PLI/systemc/osci21v1/lib-linux_gcc3_2
-L<VERDI_HOME>/share/FsdbWriter/LINUX_GNU_32
-L<OSCI_INSTALL_DIR>/lib-linux_gcc3_2
-lfsdbSC -lnffw -lsystemc
```

SystemC Dumping Limitations

1. This library only supports pure SystemC designs.
2. Non-intuitive names, such as, *signal_0*, are shown as signal names if no signal names declared in sc_main function have been passed. For example:

```
int sc_main(int argc, int argv[]){
sc_signal<int> var1;
sc_signal<int> var2("var2");
...
```

   The name of var1 is "*signal_0*", and the name of var2 is "*var2*".

Using the SystemC Dumper to Dump SystemC Designs

The *libfsdbSC.a* library provides the dumping class and functions that let you create an FSDB file containing the values of signals as they change during the simulation. Access to the dumping class and functions are provided in the signal header file named *fsdb_trace_file.h*.

The following class and functions are provided:

Class

- fsdb_trace_file

Functions

- fsdbDumpvars
- fsdbDumpOn
- fsdbDumpOff
- fsdb_trace

To dump the signals in your SystemC design, you must modify your SystemC code with the following steps:

1. Include the header file *fsdb_trace_file.h*.

```
#include "systemc.h"
#include " fsdb_trace_file.h "
...
```

2. Create the trace file at the top level after all modules and signals have been instantiated.

```
fsdb_trace_file *my_trace_file;
my_trace_file = new fsdb_trace_file("dump.fsdb");
```

3. Invoke the dumping functions to dump module instances or signals explicitly.

```
fsdbDumpvars(my_trace_file)
```

4. Link the *libnffw.a* and *libfsdbSC.a* libraries.

   You can find *libnffw.a* in the *<VERDI_HOME>/share/FsdbWriter/$PLATFORM* sub-directory.

Linking the Novas Library with the Open SystemC Initiative Simulator

NOTE: Linux and gcc2.96 are used as examples in the following steps.

To build a SystemC executable that contains the Novas PLI application functions, use the following steps:

1. Set the library path.

```
setenv LD_LIBRARY_PATH
<VERDI_HOME>/share/FsdbWriter/LINUX_GNU_296: $LD_LIBRARY_PATH
```

2. Compile your design. You should specify the compiler option *-I<VERDI_HOME>/share/PLI/systemc/<sc_version_dependant>/include.*

```
g++ -c *.cpp
-I<VERDI_HOME>/include
-I<VERDI_HOME>/share/PLI/systemc/include
-I<VERDI_HOME>/share/PLI/systemc/old/osci201/include
```

3. Build an executable file. You should link the *libfsdbSC.a* and *libnffw.a* libraries.

```
g++ *.o -o run.x
-L<VERDI_HOME>/share/PLI/systemc/old/osci201/lib-
linux_gcc2_96
-L<VERDI_HOME>/share/FsdbWriter/LINUX_GNU_296
-L<OSCI_INSTALL_DIR>/lib-linux_gcc2_06
-lfsdbSC -lnffw -lsystemc
```

Data and Object Types Supported by the SystemC and C Programming Languages

In the simulator IUS 13.1, VCS MX 2014.03, ModelSim SE 10.2 and their later versions, the System C and C programming languages support the following

object and data types: sc_signal<T>, sc_buffer<T>, and sc_in<T>/sc_out<T>/sc_inout<T>. The data types can be inserted as the <T> in the object types.

NOTE: If the dumper option is not specified, the sc_signal<T>, sc_buffer<T>, and sc_in<T>/sc_out<T>/sc_inout<T> types will be dumped automatically.

Supported Data Types

Language	Data Type
SystemC	sc_bit
	sc_logic
	sc_bv<w>
	sc_lv<w>
	sc_int<w>
	sc_uint<w>
	sc_signed
	sc_unsigned
	sc_fxval
	sc_fxval_fast
	sc_fxnum
	sc_fxnum_fast
C	char
	unsigned char
	short
	unsigned short
	int
	unsigned int
	long
	unsigned long
	long long
	unsigned long long
	double

For the resolved signal, the System C program language supports the object types in the simulator IUS 13.1, VCS MX 2014.03, ModelSim SE 10.2 and their later versions.

SystemC Supported Object Types

Simulator Version	Object Type
IUS 13.1 and later version	sc_signal_rv<w>
	sc_in_rv<w>
	sc_inout_rv<w>
	sc_out<w>
	sc_logic_resolved
	sc_in_resolved
	sc_inout_resolved
	sc_out_resolved
VCS MX 2014.03 and later version	sc_signal_rv<w>
	sc_in_rv<w>
	sc_logic_resolved
	sc_in_resolved
	sc_inout_resolved
ModelSim SE 10.2 and later version	sc_signal_rv<w>
	sc_in_rv<w>
	sc_logic_resolved
	sc_in_resolved
	sc_inout_resolved

Appendix A: Third Party Integration

Linking With Synopsys VCS Simulators (VCS 2015.09 or Later Versions)

Using the Stackable -P Option for Specifying Multiple Libraries

When using the Novas object files along with custom PLI libraries (which may use the FSDB library, *FsdbWriter*), refer to *<VERDI_HOME>/demo/dumper/vcs_link_third_party* for a detailed example.

Linking with Cadence Simulators (IUS 13.1 or Later Versions)

Refer to the *<VERDI_HOME>/demo/dumper* directory for detailed examples of the different methods to link the *Cadence* simulator and other third party integrations with the Novas object files for FSDB dumping.

Verilog Only

Dynamically Link Novas Object File With Other Applications

Use one of the following methods to dynamically link IUS with the Novas object files for FSDB dumping alone or with other applications. C code is used to demonstrate how to integrate the Novas object files for FSDB dumping with a third party application or library.

Method 1 - Bootstrap Function

If the other applications support bootstrap functions for dynamic loading, the `-loadpli1` option can be used with `ncelab` (or the `+loadpli1` option with `ncverilog`) to load them and then dynamic linking can be used to load the `libpli.so` (`libpli.sl`) shared library of the Novas object files for FSDB dumping by default. If the third-party's application or library supports bootstrap method, it should contain its own boot function (its own `veriuser.c` in binary format). Refer to the *<VERDI_HOME>/demo/dumper/vlog_pli_dynamic_boot* directory for a detailed example.

NOTE: The FSDB dumper has no dependency on any other symbol. If a third party application has the dependency of symbols located in the FSDB dumper, use the "`:export`" qualifier as shown below when loading the FSDB Dumper, or refer to the simulator's manual regarding `-pli_export` for details.

```
> ncverilog +ncaccess+r -f run.f\
+loadpli1=debpli:novas_pli_boot:export \
+loadpli1=<third_party_PLI_path>:<boot_fn>:expo
rt
```

or

```
> ncelab -access +r -f ncelab.args\
-loadpli1 debpli:novas_pli_boot:export \
-loadpli1 <3rd_party_PLI_path>:<boot_fn>:export
```

Instead of adding the :export qualifier, you can use the -pli_export option. For example:

```
> ncverilog +ncaccess+r +ncpli_export -f run.f\
+loadpli1=debpli:novas_pli_boot \
+loadpli1=<third_party_PLI_path>:<boot_fn>
```

or

```
> ncelab -access +r -pli_export -f ncelab.args\
-loadpli1 debpli:novas_pli_boot \
-loadpli1 <third_party_PLI_path>:<boot_fn>
```

Method 2 - Single Shared Library

The Novas object files for FSDB dumping can be merged with other applications to become a single shared library. A custom `veriuser.c` file can be merged with the one that the FSDB dumper provides, and archive the merged modified `veriuser.o` custom sample binary and FSDB dumper object file together to become a single shared library. After that, there is only one single shared library that needs to be linked while calling the FSDB dumping commands or the custom

task commands. Refer to the *<VERDI_HOME>/demo/dumper/vlog_custom_pli* directory for a detailed example.

NOTE: For cases where making standalone `ncelab` and `ncsim` executable object is desired, try with the `make -f Makefile.nc static` method as addressed in the example.

Method 3 – Single CFC Shared Library

The CFC library is specifically for commands called under Tcl command prompt, `ncsim>`. This method merges the custom CFC library with the Novas CFC library to become a single shared library, `libcfc.so`. A custom `cfc_user.c` file can be merged with the `novascfc.c` file that the FSDB dumper provides, and archive the merged modified `novascfc.o`, custom sample binary and FSDB dumper object file together to become a single shared library, `libcfc.so`. After that, there is only one single share library that needs to be linked while calling the FSDB dumping commands or the custom task commands under the simulator command prompt, `ncsim>`. Refer to the *<VERDI_HOME>/demo/dumper/vlog_cfc_pli* directory for a detailed example.

NOTE: For the situation where making standalone `ncelab` and `ncsim` executable objects is desired, try the `make -f Makefile.nc static` method as addressed in the example.

Static Linking

The Novas object files for FSDB dumping can be statically linked with the IUS executable files, `ncelab` and `ncsim`. If this method is selected, new `ncelab` and `ncsim` executable files need to be built when a different release of IUS or the Verdi platform is used. Refer to the *<VERDI_HOME>/demo/dumper/vlog_static_link* directory for a detailed example.

VHDL Only

In pure VHDL designs, the FMI library is required for commands specified inside the VHDL design and the CFC library is required for commands called under the Tcl command prompt in interactive mode. Third-party CFC and FMI can be merged into the FSDB dumper's CFC and FMI libraries. Refer to the *<VERDI_HOME>/demo/dumper/vhdl_cfc_fmi* directory for a detailed example.

Mixed Verilog/VHDL

In mixed language designs, the FMI library is required for commands specified inside the VHDL design, the PLI library is required for commands specified inside the Verilog design, and the CFC library is required for commands called under the Tcl command prompt in interactive mode. Third-party PLI, CFC, and FMI can be merged into the FSDB dumper's PLI, CFC, and FMI libraries. Refer to the *<VERDI_HOME>/demo/dumper/mixed_cfc_fmi_pl*i directory for a detailed example.

Linking With ModelSim Simulators (10.2 or Later Versions)

Build a New Shared Library to Include Other Applications

You may want to load just one shared library into *vsim* that contains the Novas object files for FSDB dumping and other applications (your own or other vendors). Refer to the *<VERDI_HOME>/demo/dumper/modelsim_link_third_party* for a detailed example.

Appendix B: Required VCS MX Options for Linking With FSDB Dumper

The following table is a quick reference of required VCS options when linking with FSDB dumping. It is based on VCS MX 2013.06 and is valid until VCS MX 2014.12 version.

NOTE: The list below is subject to change without notice due to changes in different simulator versions. Refer to the appropriate simulator manual for the latest information.

Affected Command -line Executable	Option or Setting	Requirement	Related FSDB Dumping Commands
vcs	+vcsd	Required for all FSDB commands related to dumping value change data of any kind.	$fsdbDumpvars, $fsdbDumpMDA, $fsdbDumpSVA, $fsdbDumpvarsByFile ($fsdbDumpvarsToFile)
vcs	+vpi	Required for all FSDB commands related to dumping value change data of any kind.	$fsdbDumpvars, $fsdbDumpMDA, $fsdbDumpSVA, $fsdbDumpvarsByFile ($fsdbDumpvarsToFile)
vcs	+memcbk	Required for FSDB dumping command where MDA/ memory is desired.	$fsdbDumpMDA, $fsdbDumpvars +mda

Affected Command -line Executable	Option or Setting	Requirement	Related FSDB Dumping Commands
vcs	-ucli	Required for UCLI mode (interactive "ucli%" command prompt). The additional "-load libnovas.so: FSDBDumpCmd" option is also required. **NOTE: -debug_access+pp/ -debug_access/- debug_access+all** should cover this.	All FSDB commands in “ucli%” command prompt: > simv -ucli ucli% fsdbDumpvars 0 or > simv -ucli -i cmd.do cmd.do file contains fsdbDumpvars 0
vcs	-debug_access+pp	Optional; equivalent to an aggregation of **+vcsd +vpi +memcbk +cli+1** and **-ucli**.	All
vcs	-debug_access	Optional; equivalent to an aggregation of **-debug_access+pp** plus **+cli+4**; able to force net and reg, set value, time break.	All
vcs	-debug_access+all	Optional; equivalent to **-debug_access** plus line stepping, debugging; required by dumping variables under VHDL process.	All (specifically for fsdbDumpvars +trace_process)
vcs	-rdynamic	Required for situations where environments (scripts) are bound with the OLD FSDB dumper (or FsdbWriter) in static linking and cannot be removed. **NOTE:** Valid for LINUX platform.	N/A

Affected Command -line Executable	Option or Setting	Requirement	Related FSDB Dumping Commands
vcs	-P $VERDI_ HOME/shared/PLI/ VCS/ $PLATFORM/ novas.tab \ $VERDI_HOME/ shared/PLI/VCS/ $PLATFORM/pli.a	Required for all Verilog/SV FSDB dumping.	All
vcs	-load libnovas.so: FSDBDumpCmd	Required for all FSDB dumping commands in UCLI mode entry. **NOTE:** Valid for pure Verilog/SV or mixed (Verilog/SV and VHDL) scenarios.	All
vcs	-vhpi novas:FSDBDump Cmd	Required for all FSDB dumping commands in UCLI mode entry. **NOTE:** Valid for pure VHDL scenarios.	All
vcs	-fsdb/-fsdb_pp	For smooth migration of conventional "-fsdb" users to adopt the new FSDB dumper. **NOTE**: 1. Valid since VCS2013.06 version. 2. "*VERDI_HOME*" environment variable is required to be set to the Verdi installation directory. 3. **-fsdb_new:** also set VCS debug option: "**+cli+3**". **-fsdb_pp_new:** also set VCS debug option: "**+vcsd+cli+1**".	All
simv	+fsdb+parallel= on	To enable parallel FSDB dumping with all VCS versions.	All

Affected Command-line Executable	Option or Setting	Requirement	Related FSDB Dumping Commands
simv	setenv LD_LIBRARY_PATH ${VERDI_HOME}/share/PLI/VCS/${PLATFORM}	Required for loading FSDB dumper's core library, VHDL procedure call entry library, or UCLI command entry library.	All

Appendix C: Dumping to Multiple Files with FSDB Commands

Overview

The FSDB dumper provides a more flexible method to have multiple dumping files opened to collect different dumping data in a simulation run. With simple rules, each dumping file can be configured with different combinations, such as dumping scopes, dump off time or dump on time, dumping tasks, and options. By using dumping tasks and the scheme of multiple files, dumping multiple specified modules into different files can be easily achieved.

+fsdbfile+ Aware Dumping Commands

The following dump commands can take effect with the `+fsdbfile+` option:

- `$fsdbDumpvars`
- `$fsdbDumpMDA`
- `$fsdbDumpSVA`
- `$fsdbDumpvarsByFile`
- `$fsdbDumpoff`
- `$fsdbDumpon`
- `$fsdbDumpfile`
- `$fsdbAutoSwitchDumpfile`

NOTE: `$fsdbDumpfile` and `$fsdbAutoSwitchDumpfile` do not require the `+fsdbfile+` prefix.

Default FSDB Files and Target FSDB Files

The first created FSDB file that is opened for collecting simulation data is defined as the default FSDB file.

```
$fsdbDumpfile("my_first");
$fsdbDumpvars("+all");
```

- The first command specifies the default file name as `my_first.fsdb`.
- The second command performs `dumpvars +all` to `my_first.fsdb`.

If more than one FSDB file is opened for dumping, the `+fsdbfile+target` option is used to distinguish the affected FSDB files.

```
$fsdbDumpfile("my_1st");
$fsdbDumpfile("my_2nd");
$fsdbDumpvars(0, top.inst1, "+fsdbfile+my_2nd");
```

- The first command specifies the default file name as `my_1st.fsdb`.
- The third command dumps `top.inst1` to the target FSDB file `my_2nd.fsdb`.
- The `my_1st.fsdb` file contains nothing.

Subsequent dumping tasks without specifying the `+fsdbfile+target` option affects the default FSDB file.

```
$fsdbDumpfile("my_1st");
$fsdbDumpfile("my_2nd");
$fsdbDumpvars(top.inst1, "+fsdbfile+my_2nd");
$fsdbDumpvars(top.inst2);
```

- The first command specifies the default file name as `my_1st.fsdb`.
- The third command dumps signals under the scope `top.inst1` to the target FSDB file `my_2nd.fsdb`.
- The fourth command dumps signals under the scope `top.inst2` to the default FSDB file `my_1st.fsd`*b*.

The default FSDB file name can either be specified with the first `$fsdbDumpfile` or `+fsdbfile+filename` task argument, or be given by the default file name, *novas.fsdb*.

For `$fsdbDumpoff` and `$fsdbDumpon` commands, if the target FSDB file is not specified, all opened FSDB files in the current simulation is affected.

Usage Examples

Example 1

```
$fsdbDumpvars("+all");
$fsdbDumpMDA("+fsdbfile+my_mda_file");
$fsdbDumpSVA();
```

1. The first command creates the default file (the file name is `novas.fsdb`) and performs `dumpvars +all` to the `novas.fsdb` file.
2. The second command dumps MDAs to the `my_mda_file.fsd`*b* file.
3. The third command dumps SVA to the default file `novas.fsdb`.

Example 2

```
$fsdbDumpvars(top.inst1, "+fsdbfile+my_inst1");
$fsdbDumpvars(top.inst2, "+fsdbfile+my_inst2", "+all");
$fsdbDumpMDA(top.inst1);
$fsdbDumpSVA(top.inst2, "+fsdbfile+my_inst2");
```

1. The first command specifies the default file name as `my_inst1.fsdb`.
2. The first command specifies to dump signals under the scope `top.inst1` to the `my_inst1.fsdb` file.
3. The second command dumps all under `top.inst2` to `my_inst2.fsdb`.
4. The third command also dumps MDAs under `top.inst1` to the default file.
5. The fourth command dumps SVA under `top.inst2` to `my_inst2.fsdb`.

Example 3

```
$fsdbDumpvars(top.inst1, "+fsdbfile+my_inst1", "+all");
$fsdbDumpvars(top.inst2, "+fsdbfile+my_inst2", "+all");
#100 $fsdbDumpoff("+fsdbfile+my_inst1");
#100 $fsdbDumpoff("+fsdbfile+my_inst2");
#100$fsdbDumpon();
```

1. The first command specifies the default file name as `my_inst1.fsd`*b*.
2. The first command dumps all under `top.inst1` to `my_inst1.fsdb` and the second command dumps all under `top.inst2` to `my_inst2.fsd`*b*.
3. `my_inst1.fsdb` performs dump off at simulation time 100 and `my_inst2.fsd`*b* performs dump off at time 200.
4. At simulation time 300, both `my_inst1.fsdb` and `my_inst2.fsdb` perform dump on, because the `$fsdbDumpon` command specifies no specific target file, and all opened FSDB files in the current simulation is affected.

Example 4

```
$fsdbAutoSwitchDumpfile(10, "my_full_dump", 0);
$fsdbDumpvars("+all");
$fsdbDumpSVA("+fsdbfile+my_sva");
```

1. The first command specifies the default file name as `my_full_dump.fsd`*b*.
2. The second command dumps all to the `my_full_dump.fsdb` file.
3. The third command dumps SVA to `my_sva.fsdb`.
4. Whenever file size reaches 10 MB, switching files automatically takes effect on the `my_full_dump.fsdb` file.

Appendix D: Switch Dumping With Simulator Restore

The FSDB dumper supports switching the FSDB dump file when the `restore` command of the simulator is executed. When the `restore` (or `restart`) command of the simulator is executed for the first time, the FSDB dumping is redirected to a new FSDB file named as `<original_FSDB_name>.restore.FSDB`. The original dumping files are saved and closed. If multiple FSDB files are being dumped, each of them is redirected to a different restore FSDB file. When the `restore` command of the simulator is executed several times, the *<original_FSDB_name>.restore.FSDB* files are overridden.

NOTE: For VCS simulator, new restore file is not created after the `restore` command of the simulator is executed.

Synopsys VCS Simulator

When the `restore` command is executed for the first time, the dumping is saved to the original dump file. If multiple FSDB files are dumped, all of them are saved to the original dump file.

Simulator Commands

- save
- restore

Refer to the *VCS User's Manual* for details of `save` and `restore` commands.

NOTE: The `VCS_ENABLE_ASLR_SUPPORT` environment variable must be set to 1 before running the simulation. For example:
`> setenv VCS_ENABLE_ASLR_SUPPORT 1`

Example

```
ucli> fsdbDumpvars {"+fsdbfile+save.fsdb"}
```

Create the `save.fsdb` FSDB file.

```
ucli> run 5ns
ucli> save saveData
```

Save a copy of the simulation data at the time for later restore.

```
ucli> run 20ns
ucli> restore saveData
```

When the simulation is restored to the previously saved time point, the `save.fsdb` file is restored to 5ns.

```
ucli> run 30ns
ucli> exit
```

The following FSDB file and associated time range is the result:

- save.fsdb: 0ns ~ 35ns

NOTE: User-defined Tcl procedures that were loaded before saving are not saved when executing the `save` command. To use these Tcl procedures after the restore process, load these Tcl procedures again.

ModelSim Simulator

When the `restore` command is executed for the first time, the dumping is redirected to a new FSDB file named as `<original_FSDB_name>.restore.FSDB`. If multiple FSDB files are being dumped, each of them is redirected to a different restore FSDB file.

Simulator Commands

- checkpoint
- restore

Refer to the *ModelSim User's Manual* for details of `checkpoint` and `restore` commands.

Example

```
vsim> fsdbDumpvars +fsdbfile+save.fsdb
```

Create the `save.fsdb` FSDB file.

```
vsim> run 10ns
vsim> checkpoint saveData
```

Save a copy of the simulation data at the time for later restore.

```
vsim> run 20ns
vsim> restore saveData
```

When the simulation is restored to the previously saved time point, the `save.fsdb` file is saved and closed, and the new file `save.restore.fsdb` is created.

```
vsim> run 30ns
vsim> quit
```

The following FSDB files and associated time ranges are the result:

- save.fsdb: 0ns ~ 30ns
- save.restore.fsdb: 10ns ~ 40ns

Cadence IUS Simulator

When the `restart` command is executed for the first time, the dumping is redirected to a new FSDB file named as `<original_FSDB_name>.restore.FSDB`. If multiple FSDB files are being dumped, each of them is redirected to a different `restore` FSDB file.

Simulator Commands

- save
- restart

Refer to the IUS user's manual for details of the `save` and `restart` commands.

Example

```
ncsim> call fsdbDumpvars +fsdbfile+save.fsdb
```

Create the `save.fsdb` FSDB file.

```
ncsim> run 20ns
ncsim> save saveData
```

Save a copy of the simulation data at the time for later restore.

```
ncsim> run 20ns
ncsim> restart saveData
```

When the simulation is restored to the previously saved time point, the `save.fsdb` file is saved and closed, and the new file `save.restore.fsdb` is created.

```
ncsim> run 10ns
ncsim> exit
```

The following FSDB files and associated time ranges are the result:

- save.fsdb: 0ns ~ 40ns
- save.restore.fsdb: 20ns ~ 30ns

Miscellaneous Rules

The miscellaneous rules apply in the following situations:

- When the `restore` command is executed several times.
- When the FSDB name happens to have a `restore.fsdb` extension.
- When restoring in a new simulation session.

restore Command Executed Multiple Times

When the `restore` command is executed several times, the `<original_FSDB_name>.restore.FSDB` files are overridden.

Example in IUS

```
ncsim> call fsdbDumpvars +fsdbfile+save.fsdb
```

Create the `save.fsdb` FSDB file.

```
ncsim> run 20ns
ncsim> save saveData
```

Save a copy of the simulation data at the time for later restore.

```
ncsim> run 20ns
ncsim> restart saveData
```

When the simulation is restored to the previously saved time point, the `save.fsdb` file is saved and closed, and the new file `save.restore.fsdb` is created.

```
ncsim> run 10ns
ncsim> restart saveData
```

When the simulation is restored to the previously saved time point, the `save.restore.fsdb` file is recreated and this file overwrites the previous one.

```
ncsim> run 40ns
ncsim> exit
```

FSDB File with *restore.fsdb* Extension

When the FSDB name happens to have a `restore.fsdb` extension, then the restore FSDB file name is the same.

Example in VCS

```
ucli> fsdbDumpvars {"+fsdbfile+save.restore.fsdb"}
```

Create the `save.restore.fsdb` FSDB file.

```
ucli> run 5ns
ucli> save saveData
```

Save a copy of the simulation data at the time for later restore.

```
ucli> run 20ns
ucli> restore saveData
```

When the simulation is restored to the previously saved time point, the `save.restore.fsdb` file is recreated and this overwrites the previous one.

```
ucli> run 30ns
ucli> exit
```

Restore in a New Simulation Session

For *IUS* and *Modelsim*, restoring in a new simulation session is an independent dumper process. For *VCS*, the environment is exactly the same as in a single session.

Example in IUS

- The first simulation session:

  ```
  ncsim> call fsdbDumpvars +fsdbfile+save.fsdb
  ```

  The `save.fsdb` file is created.

  ```
  ncsim> run 20ns
  ncsim> save saveData
  ncsim> run 20ns
  ncsim> restart saveData
  ```

  The `save.restore.fsdb` file is created.

```
ncsim> run 10ns
ncsim> exit
```

- The second simulation session:

```
ncsim> restart saveData
ncsim> call fsdbDumpvars +fsdbfile+save.fsdb
```

The `save.fsdb` file is created.

Example in VCS

- The first simulation session:

```
ucli> fsdbDumpvars {"+fsdbfile+save.fsdb" "+all"}
```

Create the file `save.fsdb`.

```
ucli> run 5ns
ucli> save saveData
ucli> run 30ns
ucli> exit
```

- The second simulation session:

```
ucli> restore saveData
```

The `save.restore.fsdb` file is created with the `+all` option.

```
ucli> run 20ns
ucli> exit
```