

Xcelium Save/Restart

Product Version 17.04

April 2017

© 2017 Cadence Design Systems, Inc. All rights reserved.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522. All other trademarks are the property of their respective holders.

Restricted Permission: This publication is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Cadence. Unauthorized reproduction or distribution of this publication, or any portion of it, may result in civil and criminal penalties. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. Unless otherwise agreed to by Cadence in writing, this statement grants Cadence customers permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used only in accordance with a written agreement between Cadence and its customer.
2. The publication may not be modified in any way.
3. Any authorized copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement.
4. The information contained in this document cannot be used in the development of like products or software, whether for internal or external use, and shall not be used for the benefit of any other party, whether or not for consideration.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

Contents


1	5
Preface	5
Other Sources of Information	5
Typographical Conventions	5
Viewing Documentation with the Cadence Help Tool	7
Customer Support	7
Cases	7
Using Cadence Online Support	8
2	10
Overview	10
Terms	10
Usage	10
Iterative Debugging	10
Fault Tolerance	11
Common Simulation Sequence	11
Enabling Process-Based Save/Restart	11
Command Line Option	11
Environment Variables	12
Restart -process <snapshot>	12
Save	12
Invoking a process-based save	12
Save Inclusions	13
Restart	13
Warm Restart	14
Cold Restart	15

Preface

This preface contains the following sections:

- [Other Sources of Information](#)
- [Typographical Conventions](#)
- [Viewing Documentation with the Cadence Help Tool](#)
- [Customer Support](#)
 - [Cases](#)
 - [Using Cadence Online Support](#)

Other Sources of Information

 This document contains information specific to embedded software debugging.

Typographical Conventions

This and all Cadence manuals use visual cues to help you locate and interpret information easily. These cues are explained in Table 1-1.

Table 1-1: Document Conventions

Typeface	Represents
courier font	Indicates code. For example: <pre>do burst_response keeping {</pre>
courier bold	Used to highlight important sections of code, like actions. For example: <pre>do burst_response keeping {</pre>

bold	<p>The bold font indicates keywords in descriptive text. For example, the following sentence contains keywords for the show ini command and the get_symbol() routine:</p> <p>You can display these settings with the show ini setting command or retrieve them within e code with the get_symbol() routine.</p>
<i>italic</i>	<p>The italic font represents user-defined variables that you must provide. For example, the following line instructs you to type the "write cover" as it appears, and then the actual name of a file:</p> <p>write cover <i>filename</i></p>
[] square brackets	<p>Square brackets indicate optional parameters. For example, in the following construct the keywords "list of" are optional:</p> <p>var name : [list of] <i>type</i></p>
[] bold brackets	<p>Bold square brackets are required. For example, in the following construct you must type the bold square brackets as they appear:</p> <p>extend enum-type-name : [name,...]</p>
	<p>The pipe character indicates alternative syntax or parameters. For example, the following line indicates that either the bits or bytes keyword should be used:</p> <p>type scalar-type (bits bytes : <i>num</i>)</p>
{ } Braces	<p>Used with OR-bars and enclose a list of choices from which you <i>must</i> choose one.</p> <p><code>command {argument1 argument2 argument2}</code></p>
<i>construct</i> , ...	<p>An item, followed by a separator (usually a comma or a semicolon) and an ellipsis is an abbreviation for a list of elements of the specified type. For example, the following line means you can type a list of zero or more names separated by commas.</p> <p>extend enum-type-name : [<i>name,...</i>]</p>
... Three dots	<p>Indicate that you can repeat the previous argument. If they are used within brackets, you can specify zero or more arguments. If they are used outside of brackets, you must specify at least one argument, but you can specify more.</p> <p><i>argument</i> ... specify at least one [<i>argument</i>]... you can specify zero or more</p>

Viewing Documentation with the Cadence Help Tool

If `<install_dir> /tools/bin` is in your path, type:

```
cdnshelp &
```

otherwise, type:

```
<install_dir> /tools/bin/cdnshelp &
```

To open help for a specific platform, type:

```
cdnshelp <install_dir> /doc/xmlreg/ <platform> library.lbr
```

Customer Support

Your [Cadence Online Support](#) account lets you download releases of Cadence products, search for solutions to common problems, receive announcements on product releases, submit service requests, and track your open service requests.


If you have a problem using the Embedded Software Debug App or the documentation, you can submit a customer case to Cadence Support. When doing so, please provide enough information about the problem so that it can be investigated efficiently. Describe the problem in full, give the version of the software you are using, and state the exact circumstances in which the problem occurs.

Cases

Cases are your way of giving feedback, asking questions, getting solutions, and reporting problems. Unless told otherwise, Cadence Support staff will respond to your case. If Cadence support cannot answer your question, Cadence research and development personnel will get involved.

It is important to specify the severity level of the service request as accurately as possible. There are three levels of severity:

- **Critical** – You cannot proceed without a solution to the issue.
- **Important** – You can proceed, but you need a solution to the issue.
- **Minor** – You prefer to have a solution, but you can wait for it.


 You can request Support to increase the severity level of an issue. Therefore, do not use **Critical** unless resolution of an issue is absolutely necessary and urgently required.

Using Cadence Online Support

Cadence encourages you to submit cases using Cadence Online Support. With Cadence Online Support you can also track your open cases.

To use Cadence Online Support to submit a case:

1. If you do not yet have a Cadence Online Support account, go to [Cadence Online Support](#) and click *Register Now* under the *New User* heading. You must provide a valid *HostID* for any Cadence product. The *HostID* is contained in the SERVER line of your Cadence product license file.

 If you already have a Cadence Online Support account, then you only need to update your Cadence Online Support preferences to include a valid HostID for a Cadence product.

2. Log in to Cadence Online Support, and on the upper left side of the page click *Create Case* under the *Cases* heading. A form is presented for submission of your service request. Select your product in the *Product* list box and click *Continue* . Follow the online instructions to complete the Service Request.

Creating Group Privileges in Cadence Online Support

Sometimes it is beneficial to view the cases of others on your project.

To create group privileges in Cadence Online Support:

1. Open a Cadence Online Support service request by clicking *Create Case* under the *Cases* heading.
2. Select your product in the *Product* list box and click *Continue* .
3. Fill in the required fields in the form presented. Explain in the Stated Problem text box that you want to create a group of users.
4. In the *People to notify upon Case creation* field, include the email addresses of the users you

want to have group privileges.



Each person receiving group privileges must have a Cadence Online Support account.

5. Click *Submit Case* to complete the case.

Visit the [Cadence Support home page](#) for an overview of customer support offerings and the support process.

Overview

Process-Based Save/Restart for Xcelium saves the entire state of the simulation process, including all file descriptors.

Terms

Saving: Saves the simulation snapshot. This feature saves the state of the DUT and the verification environment.

Restarting: Loads the state of the simulation snapshot back into the simulator. This feature restarts a simulation from the time where the save function occurred.

Reseeding: Changes the seed for randomization operations. Instead of using the original seed for multiple randomizations, a new seed is used.

Usage

Restart options include "warm" and "cold" restarts, which provide different uses with your simulation.

Iterative Debugging

You can create a checkpoint of the simulation state before an error situation requiring iterative interactive debugging. Typically, you would perform a "warm restart" to allow for multiple debug passes, with a save state near where debugging is necessary. A warm restart allows you to repeatedly simulate from that save to debug, selectively turning on the collection of useful debugging information. You can repeat debugging in order to capture enough debug information. Upon warm restart (from the Tcl prompt), the simulation state is restored to exactly the same state as the saving process including GUI status, simulator debug state, file descriptors, and third-party C code.

Fault Tolerance

A cold restart allows for fault tolerance, typically for long multi-day simulations. In this case, periodic saves allow you to recover in case of issues, such as a server or system fault. You can periodically create checkpoints of the simulation, overwriting any previous checkpoints. The simulation can be restarted from that checkpoint to continue the simulation, without needing to re-simulate behavior before the checkpoint and so completing simulation more quickly than otherwise would be possible. Upon cold restart, specific command-line options are available to enable you to run a different test scenario from the saved state. Running in a different directory than the saving simulation is also supported.

Common Simulation Sequence

Save/restart provides you with a common simulation sequence, for example, an SOC boot initialization sequence, among multiple regression tests. In this scenario, the common initialization sequence is simulated once and then multiple tests are run afterward. This requires that there be a way to change some aspect of the simulation, such as changing the program executed by a software-driven test, to effect the new test. This approach saves computing resources and increases turn-around time during test development, or when debugging issues with a particular test. A cold restart can also allow for multiple tests with a shared initialization. In this case, you can run a long initialization sequence and save, and then restore multiple copies of this save. You can reseed or run new tests in each copy on the same or different systems.

Enabling Process-Based Save/Restart

Process-Based Save/Restart is enabled through either a command line option or through an environment variable. When enabled, a new `-process` option is available on both the save and restart Tcl commands.

Command Line Option

When using `xrun`, the `-checkpoint_enable` flag can be passed to `xrun` and must be used when building (elaborating) the design. When using three-step compilation, `-checkpoint_enable` should be passed to both `xmelab` and `xmsim`. This option setting is required for both the simulation session performing the save as well as any simulation session restarting the saved snapshot created with the `-process` option.

Environment Variables

If the environment variable CHECKPOINT_ENABLE is set to any value, then Process-Based Save/Restart is enabled, allowing the use of the -process option to the save command. The variable behavior is identical to the command line option.

Also, if the environment variable CHECKPOINT_DEFAULT is set, then all save and restart Tcl commands behave as if the -process option is specified. This can be useful when modifying invocation, or if Tcl scripts are difficult or impossible.

Restart -process <snapshot>

When Process-Based Save/Restart is enabled, a new -process option is added to the restart Tcl command. When this option is present, the library is searched for a process-based snapshot with the specified name. If found, a warm restart is performed.

As described above, if the environment variable CHECKPOINT_DEFAULT is set, then the -process option is implied.

Save

This section describes how to invoke a process-based save.

Invoking a process-based save

A process-based save can occur from the Tcl prompt or by using the \$save() system task.

Save -process <snapshot>

When process-based save/restart is enabled, a new -process option is added to the save Tcl command. When this option is present a process-based save is invoked. The <snapshot> argument is interpreted in exactly the same way as in the traditional save implementation. If the environment variable CHECKPOINT_DEFAULT is set, then the -process option is implied.

\$save("<snapshot>")

The system task, \$save() can be used for a process-based save. The argument used with the \$save task is a string containing the name of the snapshot. This string is interpreted in exactly the same way as the snapshot argument to the Tcl command described above. If a snapshot of the same name has been previously been saved, it is overwritten.

The -checkpoint_enable flag must have been used directly with xmelab, or indirectly through xrun, for \$save to be supported.

Save Inclusions

The table below lists what is included in the save.

Process memory state	The entire memory state of the process is saved into the library under the given snapshot directory in a <code>savedir</code> directory.
General Files	The state of all file descriptors that are open after all pre-save callbacks have been called are noted including the read/write status and current position in the file. All files are then flushed and their current state is copied into the save directory.
SHM Databases	SHM databases are also flushed, closed, and copied in to the save directory.
FSDB Databases	FSDB databases are saved and copied to the save directory.

Restart

There are two types of restarts.

- **Warm restart:** When a snapshot is restarted from inside xmsim from the Tcl prompt, it is referred to as a warm restart.
- **Cold restart:** If the snapshot is restarted from the shell command prompt using xrun or xmsim then it is referred to as a cold restart.

These restart types are explained below.

Warm Restart

A warm restart is invoked from the Tcl prompt.

Restart -process <snapshot>

When process-based save/restart is enabled, a new -process option is allowed for the restart Tcl command. When this option is present a <snapshot> argument is interpreted as a process-based snapshot and a search is made for the corresponding save directory. If one is found, then the process state is restored.

If the environment variable CHECKPOINT_DEFAULT is set, then the -process option is implied.

Simulation state

The simulation state is restored to exactly the state when the save occurred; including the state of all user 'C' applications. Unless an application does something special (non-linear file I/O, sockets, shared memory, memory-mapped I/O), no actions are required to save and restart. The simulation debug state is also restored to the same state as at the save. Any Tcl commands issued between the save and the restart that impact the simulation debug state, such as probe, stop, or deposit, are not in effect in the restarted simulation.

General Files

Since a warm restart always happens in the same directory as the save (it is running in the current working directory of the simulation) files are all available from the current working directory so the copies of the files in the save directory are not used. Each file is reopened in the same mode and position as at save time.

Log Files

Log files are specially handled during a warm restart, because data has been logged to the file that might contain useful information, such as debug output. The log file position is not set to the position of the save, rather it is set to the end of the file so that any new information is appended to the file.

Input File

If while restarting, Tcl input was coming from a file that was either an -input file, or a 'source' Tcl command, then during the restart the remaining commands after the restart command currently executing are saved in a temporary file. When the restarted simulation begins, these commands are pushed onto the Tcl stack and executed as if the process continued to read from the input file at the point immediately after the restart.

SHM Databases

Upon restart, a new transaction database file (.trn) is created with the original database. The design database (.dsn) is reused as it contains no time-dependent data. When this database is opened in SimVision, the transaction databases are merged so that continuous waveforms are presented to the user.

FSDB databases

FSDB databases are restored and moved to the point of the save. Upon replay, continuous waveforms are presented to the user.

Cold Restart

When process-based save/restart is enabled, a cold restart is executed by using xrun or xmsim and referencing a process-based save/restart snapshot on the command line using the -r option. The simulation state is restored in exactly the same way as during a warm restart documented above. The only exceptions are specified in the following sections.

General Files

If the cold restart is executed in the same directory as the save was invoked, restoring general files happens in exactly the same way as a warm restart except as modified by the command line options explained below.

If a cold restart is executed from a different directory from where the save was invoked, then the files that were copied in the save directory are first copied relative to the new current directory. Then they are treated the same as a warm restart.

SHM databases

SHM databases are treated in exactly the same manner as a warm restart, except that if the restart is executed in a different directory, then the saved .dsn and .trn files are first copied relative to the new directory.

Command Line options

On a cold restart, command line options can be used to cause different behavior in multiple cold restarts of the same saved simulation. It is this capability, combined with the ability to cold restart from a different directory that allows many parallel restarts of different tests from the same saved snapshot.

There are four types of command line options in a cold restart:

- **Unsupported:** A warning will be printed that no special action is being taken because of the setting of this option. This is currently true for the vast majority of options of simulation.
- **Match:** If a command line in the Match category is used during cold restart, the option setting and any arguments must be exactly the same as during the saving simulation.
- **Default:** If not given on the command line, then the value of this flag and any arguments are reset to be the value at the time of save. If given on the command line, the new value takes precedence. See -log and -svseed below as examples.
- **Override:** If not given on the command line, then this flag acts as if was never set independent of the value at the time of save. If given on the command line, the new value takes effect. See -input below as an example.

-Input

The -input option is treated as an override type. Any value of -input present at save time is ignored. If there is a new -INPUT argument, the new file is pushed onto the command line stack. This behavior occurs because any input file containing a 'save' command is typically followed by an 'exit' command. Restarting from a new -input file can also be used to cause completely different test or debug scenarios to be set up from the same saved snapshot. If multiple -INPUT files are used they are stacked in the order they occur on the command line. If -INPUT and -TCL are specified, then an interactive prompt appears after all -INPUT files have been processed.

-log

The -log option is treated as a default type. If no new -LOG option is specified then the old log file is restored to the position of the saved state and updates are appended. If a new -LOG option is used, the old log remains and the new log file opened. If restarted in a different directory, the original log is copied to the new directory, but if a new -logfile option is specified, a copy of the original file at the point of the save appears in the new directory, and a new log file is opened.

-svseed, +svseed=

The -svseed option is treated as a default type. The -svseed option allows a new random seed for SystemVerilog randomization to be specified. If it is used during a cold restart then the new value is used to seed random numbers. If it is not specified, then the previous value of the seed is used. The +svseed=<integer> works like -svseed. If both are present, then the value given by the -svseed option is used.

-gui

Typically the initial simulation, which is often just booting the system, is run without a GUI attached. When performing a code restart, if -gui is specified, then the cold restart simulation starts attached to a GUI.

-tcl

If the -tcl option is used, then the simulation first processes new -input files, if any, and then present a Tcl prompt for interactive input.

Plusarg handling

The state of any plusargs is a part of the simulation state and is saved. They do not need to be specified on the cold restart execution. If any plusargs are specified on the cold restart, they are prepended to the previously present plusargs. Plusargs on the restart are found before plusargs on the saving simulation. If the same plusarg is given a new value on cold restart, this is the value that will take effect. This is another mechanism to cause a cold restart to take a different simulation path than the saved simulation.