ZeBu-Server™ Release Note

Version 6_3_1

June 2011

Copyright © 2011 EVE. — All rights reserved.
This publication is confidential and may not be reproduced, in whole or in part, in any manner or in any form, without prior written permission of EVE S.A.

Table of Contents

About this document

This Release Note describes the major features available for V6_3_1 release of the ZeBu-Server software, dated 28th June 2011.

Note that the present software version is intended for ZeBu-Server only and is not intended for use with ZeBu-XL, ZeBu-XXL, ZeBu-UF or ZeBu-Personal.

This document is intended for users who are familiar with the ZeBu product range.

You can find press releases, useful white papers and technology documents on our website: http://www.eve-team.com.

It is HIGHLY recommended to read the "Installation" chapter (§ 1) and "Modified Default Settings" chapter (§ 2) before proceeding with this software release.

1 Installation

1.1 Updating the host PCs

Each host PC has to be updated by launching **`zUpdate`** (no option or specific setting) to update the PCIe interface boards. After this update, you should:

- Reboot the PCs with the `–memmap` option of the kernel.
- Launch **`zInstall`** from the V6_3_1 environment with the `-memmap` option in order to allocate the required system memory for ZeBu ↔ PC communication, as described in the *ZeBu-Server Installation Manual* (Section 6.3 in Rev. F).

Note that this update process has to be done for every host PC in a multi-unit system.

1.2 Re-generating the setup directory (`$ZEBU_SYSTEM_DIR`)

In order to upgrade your ZeBu-Server system to run with the V6_3_1 software version, you need to proceed as follows:

- Download the diagnostics patches to V 5.0 (listed in Section 7.7), whatever the software version you are upgrading from.
- Launch **`zSetupSystem`** to generate a new setup directory.
  During the **`zSetupSystem`** process, messages are displayed with instructions to update the front-panel software of the ZeBu-Server unit(s):

```
-- ZeBu : zUtils : ERROR : U0 Front-Panel software version is not up-to-date.
                           Read Vx.y vs V2.0 expected.
-- ZeBu : zUtils : ERROR : Please execute the following command :
    zUtils -loadFw U0_FP_FX $ZEBU_ROOT/etc/firmwares/ZSE/default/zse_fpcb_soft.bin

-- ZeBu : zUtils : ERROR : U1 Front-Panel software version is not up-to-date
-- ZeBu : zUtils : ERROR : Please execute the following command :
    zUtils -loadFw U1_FP_FX $ZEBU_ROOT/etc/firmwares/ZSE/default/zse_fpcb_soft.bin

-- ZeBu : zUtils : ERROR : When you have executed ALL the above commands
-- ZeBu : zUtils : ERROR : Power OFF then power ON each updated unit and execute
                           "zUtils -initSystem"
```

  - Set the `$ZEBU_SYSTEM_DIR` variable to the appropriate directory
  - Launch the commands given in the messages:

```
$ zUtils -loadFw U0_FP_FX $ZEBU_ROOT/etc/firmwares/ZSE/default/zse_fpcb_soft.bin
$ zUtils -loadFw U1_FP_FX $ZEBU_ROOT/etc/firmwares/ZSE/default/zse_fpcb_soft.bin
```

  - Power OFF/ON the ZeBu-Server units to finalize the update process.
  - Initialize the ZeBu-Server system by launching:

```
$ zUtils -initSystem
```

1.3 Compatibility with previous runtime software versions

Because of modifications of the system FPGAs in the most recent versions, once your system is initialized for V6_3_1 (**`zUtils`** `-initSystem`), you can use only the runtime software of one of the following software versions:

- V6_3_1 runtime software
- V6_3_0 with patch A_06 runtime software; any attempt to run a design with V6_3_0 without this patch will generate the following error message:

```
ZeBu : zServer : ERROR : LUI1878E : Software release you're using is not compatible
                         with the system, launch "zUtils -initSystem" with a
                         compatible release. The system has been initialized with
                         V6_3_0 version 0x00010002 whereas this software release
                         (V6_3_0) expects version 0x00010003.
```

The **`zInstall`** multi-release capability allows using runtime software of these two versions once the ZeBu-Server system has been initialized with V6_3_1.

1.4 Compatibility with designs compiled with previous software versions

It is not possible to use the V6_3_1 runtime software with a design compiled for a previous software version. If your design was compiled for V6_3_0 software, you should either:

- Use the multi-release capability of **`zInstall`** which is described in the *ZeBu-Server Installation Manual* (Section 6.2.3 in Rev. F): if **`zInstall`** is launched once for V6_3_1 and once for V6_3_0 (with patch A_06) on a system initialized for V6_3_1, the V6_3_0 runtime software is automatically launched when running a design compiled for V6_3_0.

- Save a copy of your existing compilation project (`.zpf` file) and recompile your design from scratch with V6_3_1: in **`zCui`**, set the compilation **Target** to **Project**, launch a **Clean** operation (⚙) then a **Compile** operation (⚙).

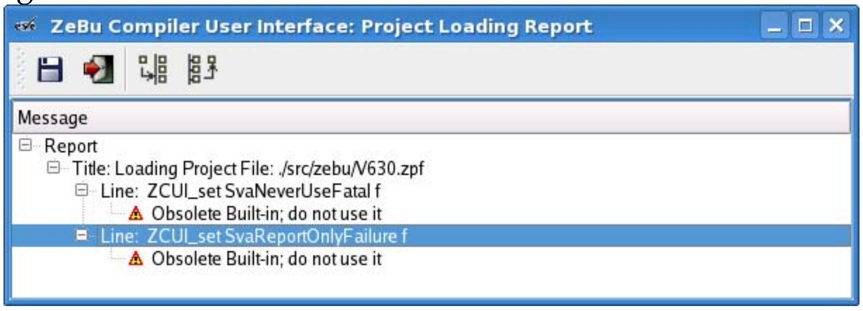
2 Modified default settings

The information in this chapter may impact the capability to compile and run a design which was previously functional with V6_3_0.

2.1 Compatibility of the `zCui` project file

Once you have saved an existing **`zCui`** project file (`.zpf`) with release V6_3_1, you can no longer use this file with previous releases. If you intend to keep your previous version project file for non-regression testing, you should save it with a different name before migrating to Version 6_3_1.

When opening an existing **`zCui`** project file, the **Report** window comes to the front and shows the problems with the project file, in particular the **`zCui`** items that have been modified and that will not be kept in the V6_3_1 project file. The following figure shows the message displayed when opening a V6_3_0 project file in which the SystemVerilog Assertions were active:



```
Report
  Title: Loading Project File: ./src/zebu/V630.zpf
    Line:  ZCUI_set SvaNeverUseFatal f
      Obsolete Built-in; do not use it
    Line:  ZCUI_set SvaReportOnlyFailure f
      Obsolete Built-in; do not use it
```

The corresponding SVA activation checkboxes have been moved in V6_3_1 from **Back-end → Debugging** tab to the **SVA** tab of each RTL group.

Note that the values for these checkboxes are set to V6_3_1 default, as described in Section 2.8 and must be checked if they were set in V6_3_0 project).

2.2 Modified names in `zCui` interface

Some panels and frames have been renamed in the present software version to improve the legibility:

- In the **Clustering** panel:
  - **Automatic zCore Generation** is now **Automatic Generation of zCores (system-level compilation)**.
  - **Performance Oriented Partitioning** is now **Performance Oriented Partitioning (zCore-level compilation)**.
- The **zCore Definition** panel has been renamed into **System-level Files** and includes the following modification:
  - **zCore Definition File** is now **Manual Declaration of zCores**.

Note that these modifications do not impact the compilation flow or the content of the corresponding files.

2.3 Modified setting in Static Timing Analysis

The model for static timing analysis has been improved, in particular for a better estimation of inter-FPGA combinational paths, inter-FPGA hops, and low-skew clock paths.

The estimated `driverClock` frequency may therefore be lower with V6_3_1 than it used to be with V6_3_0 but the capability to run at the estimated frequency is better.

2.4 Automatic setting of runtime user ID

Because the runtime user ID is now allocated automatically, as described in Section 3.7.5, manual declaration of the `ZEBU_SRD_SET_RUNID` environment variable is no longer necessary and may cause conflicts.

2.5 Modified settings for CSA

With the present software release, online CSA is disabled by default, which means that no CSA signals are processed during emulation runtime.

To have a behavior similar to V6_3_0, you can set the following environment variable:

```
$ export ZEBU_SIMULATION_SIGNAL=YES
```

The default setting for V6_3_1 is equivalent to setting this variable to `NO`.

2.6 Modified settings when dumping signals

With the previous software releases, dumping waveforms showed unexpected results: when dumping from cycle A, the number of cycles dumped in the waveform file was correct but the values of signals for cycle A were not available (data were sampled after the active edge of the clock).

This issue has been fixed in the present software release but the resulting waveform files are different and cannot be compared to waveforms dumped with V6_3_0.

To have a behavior similar to V6_3_0, the following environment variables should be set manually:

```
$ export ZEBU_DRIVER_WAVEFORM_DYNAMIC_PRE_DUMP=NO
$ export ZEBU_WAVEFORM_DYNAMIC_FORCE_INITIAL_DUMP=YES
```

However, it is not possible to have a behavior similar to V6_3_0 when dumping with the `WaveFile` class, whatever the values for these environment variables.

Note that the default setting for V6_3_1 is equivalent to (it is not necessary to set the environment variables manually, except if they were previously modified):

```
$ export ZEBU_DRIVER_WAVEFORM_DYNAMIC_PRE_DUMP=YES
$ export ZEBU_WAVEFORM_DYNAMIC_FORCE_INITIAL_DUMP=NO
```

2.7 Obsolete commands in `zDbPostProc`

The following **`zDbPostProc`** commands are no longer supported in V6_3_1:

- `close_db` (use `close_database` instead)
- `open_db` (use `open_database` instead)
- `save_db` (use `save_database` instead)
- `delete` (not replaced)

Scripts written for a previous software version may need to be modified.

2.8 New Default Mode for SVAs

When synthesizing SVAs with **`zFAST`**, the **Report Only Failure** checkbox is now selected by default.

In comparison with the full report mode which was the default in V6_3_0, this change reduces the necessary logic for SVAs and allows higher runtime performance.

To have the same behavior as with V6_3_0, you have to clear the **Report Only Failure** checkbox.

This modification impacts new **`zCui`** projects as well as any existing project, whatever the activation of SVAs and the selected options in the V6_3_0 project.

Note that this checkbox is no longer in the **Back-End** → **Debugging** tab: it has been moved to the **SVA** tab for each RTL group synthesized with **`zFAST`**.

2.9 ZEMI-3 Transactors

2.9.1 Linking with the ZeBu runtime libraries for a threadsafe environment

In previous software versions, the `libZebuZEMI3.so` library was linked with the `libZebuThreadSafe.so` library and the threadsafe environment was set by default when emulating with a ZEMI-3 transactor. It was not necessary for user to link explicitly its testbench with `libZebuThreadSafe.so` library.

Starting with V631, the `libZebuZEMI3.so` library is no longer linked with the `libZebuThreadSafe.so` library so that user can choose to use another ZeBu runtime library.
Linking explicitly the testbench with `libZebuThreadsafe.so` provides the same behavior as previous software versions.
Linking with `libZebu.so` may provide better runtime performance in some specific environments.

Such linking generally occurs in a Makefile or in a test bench compilation script.

2.9.2 Deprecated Environment Variables for WaitGroup

The `ZEMI3_EXPORTS_USE_WAITGROUP` and `ZEMI3_EXPORTS_WAITGROUP_TIMEOUT` environment variables which have been introduced in V6_2_1 for Wait Group capability have been deprecated. They are now replaced by C++ methods / C functions in the ZEMI-3 API, as described in Section 3.8.

If the environment variables are set before running the ZEMI-3 testbench, an error message is displayed:

The environment variables must be unset:

```
$ unset ZEMI3_EXPORTS_USE_WAITGROUP
$ unset ZEMI3_EXPORTS_WAITGROUP_TIMEOUT
```

3 New Features

3.1 Multi-Unit / Multi-User

3.1.1 Support of 10-unit ZeBu-Server configurations

It is now possible to interconnect up to 10 ZeBu-Server units.

Since the maximum number of users allowed on a ZeBu-Server system depends on the size of the ZeBu-Server system configuration, the maximum number of users is increased from 24 (in a system with 5 5-slot units) to 49 users (in a system with 10 5-slot units).

Detailed information about the connection for a multi-unit system and for multi-user capability is available in the *ZeBu-Server Installation Manual* (Rev. F).

3.1.2 Multi-PCIe feature

In a multi-unit configuration, up to 5 PCIe boards can be plugged in a single PC with each PCIe board connected to 1 or more dedicated ZeBu-Server units.

If your ZeBu-Server configuration includes…	then you can use:
2 units	1 or 2 PCIe boards
3 units	3 PCIe boards
4 units	4 PCIe boards
5 to 10 units	5 PCIe boards

See *ZeBu-Server Installation Manual*, Rev. F, for details on how to use **`zInstall`** in a multi-unit/multi-PCIe configuration.

3.1.3 Generating default configuration files for multi-unit systems

It is now possible to generate a default hardware configuration file with **`zConfig`** for multi-unit systems with a specific interconnection topology. This is applicable in particular for the specific topology which provides a scalable system that does not require modification of the existing cables.

The new `-topo` option which is now available for **`zConfig`** must be used simultaneously with either `-default` or `-nbfgpa` options:

```
$ zConfig -topo 84481-3 -default <system_config> -o <output_path>
```

or:

```
$ zConfig -topo 84481-3 -nbfpga <N> -o <output_path>
```

Where:

- `<system_config>` is the description of the ZeBu-Server system as described in the *ZeBu-Server Installation Manual* (Section 9.5.1 in Rev F).
- `<N>` is the number of FPGAs of the ZeBu-System (from 4 to 800 to match all possible single-unit or multi-unit systems).
- `<output_path>`: directory where `zse_configuration.tcl` is generated.

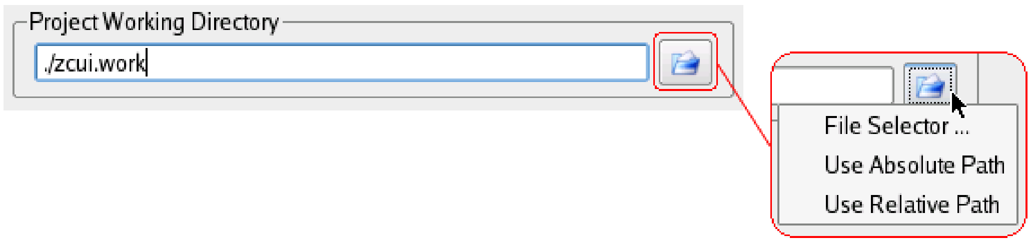
There is only one value for `-topo` option (`84481-3`) because it is the only additional topology which is supported with the present software version.

3.2 Compilation Interface (`zCui`)

3.2.1 Interface enhancements

This release introduces the following enhancements for **`zCui`** (some of the listed items were delivered with intermediate patches for V6_3_0 software version):

- All the file selectors of **`zCui`** have been modified: the 3 buttons for the file selector and for the **Absolute/Relative** paths are now gathered in a pop-up menu which is available when left-clicking an icon-button:



  Note that the items in the pop-up menu vary with the possible actions of each type of file in **`zCui`**.

- A search toolbar has been added for the **`zCui`** built-in HTML file browser with usual text search features (forward, backward, case sensitivity, search wrapped, etc).

- The following panels have been modified to take into account compilation new features:
  - In the **Clustering** panel, some frames have been created for **Automatic Generation of zCores (system-level compilation)** and for **Performance Oriented Partitioning (zCore-level Compilation)** to select and configure timing optimization, as described in Section 3.11.1.
  - When synthesizing a RTL group with **`zFAST`**:

    The **zFAST** tab has been modified to provide options for CSA in the **Accessibility vs Debugging** frame.

    In the **SVA** tab, **Never use as Fatal** and **Report only Failure** checkboxes have been added (they were previously in the **Backend → Debugging** panel).

    In the **zDPI** tab, the **Synthesize $display** checkbox has been added to support synthesis of `$display` system calls in the design, as described in Section 3.3.1.

    A new **External Synthesis** tab is available to declare additional scripts when specific modules of the design need a separate synthesizer, as described in Section 3.3.4.

3.2.2 Start & Stop Compilation

It is now possible for a user to configure in **`zCui`** which steps of the compilation flow will be processed and which will not be. This feature is useful for example when the RTL source files are expected to be modified but the user wants to work on the back-end compilation without taking into account the potential RTL modifications.
For that purpose, a dedicated view can be accessed from the **Target** selector in the **`zCui`** toolbar, by selecting the **Custom Backend: <backend-name>**. This opens a dedicated graphical interface which shows a graph of the compilation tasks of the project.
For each task in this graph, user can select one of the following options:

- compile the task (green indicator)
- keep the task as is (green indicator with keylock) and go on with compilation
- stop before the task (red indicator)

The description of this feature is not available in the *ZeBu-Server Compilation Manual*.

3.3 zFAST synthesis

3.3.1 Synthesis of `$display` system calls

When synthesizing the design with **zFAST**, `$display` system calls are now supported in the design. It is implemented as a part of the zDPI feature.
In **`zCui`**, a dedicated checkbox is available in the **zDPI** tab to activate synthesis of `$display`.
In the testbench, this `$display` system call uses the same control as the zDPI feature, as described in the dedicated *ZeBu Application Note* (AN029).

3.3.2 New DSP Allocation

When allocating DSP for multipliers, **zFAST** now uses a 25x18 multipliers based on Xilinx DSPs macros in replacement of 18x18 macros. **zFAST** also optimizes the use of 25x18 DSPs based on sub-parts of the multiplication instead of considering only the total size.
The way the **Resource Inference Threshold → DSP based multipliers** value declared in **`zCui`** is used has changed in comparison to V6_3_0 because of the above modifications. The default threshold is 16.

For example, a 32x32 multiplication is now split into in 4 sub-parts: 24x17 (41 bits), 24x15 (39 bits), 8x17 (25 bits), 8x15 (23 bits).

- With the default threshold (16), this multiplication requires 4 DSPs.
- With a threshold set to 40, this multiplication requires 1 DSP and other sub-parts are mapped onto LUTs.
- With a threshold set to 30, this multiplication requires 2 DSPs and other sub-parts are mapped onto LUTs.
- With a threshold set to 23, this multiplication requires 3 DSPs and other sub-parts are mapped onto LUTs.

3.3.3 Port Propagation

In order to reduce the amount of logic resulting from synthesis, it is now possible to activate port propagation by adding the following attribute:

```
Compile:PropagatePorts=true
```

When this attribute is set, unconnected ports of instances and constant ports of instances are propagated through the hierarchy of the design, as soon as they are explicitly unconnected or connected to constant when the source files are elaborated.

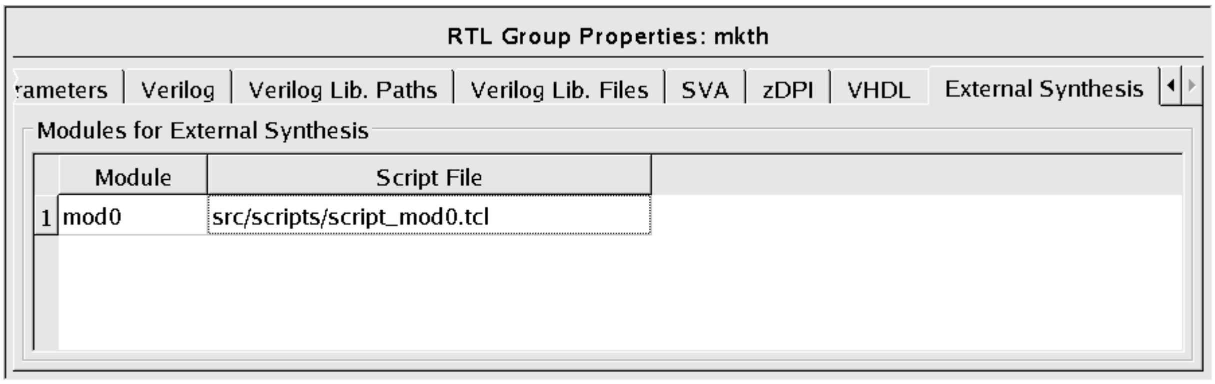
The following examples are typical syntaxes for unconnected and constant ports:

```
mod ins(.prt_unconnected());
mod ins(.prt_constant(1'b1));
```

3.3.4 External Synthesis

When a design is synthesized with **zFAST**, some modules may require to be synthesized by a third-party synthesizer (in particular some specific DesignWare components which would not be supported by ZW-FPGA). In previous software versions, it was necessary to synthesize such modules out of **zCui** and use EDIF source files.

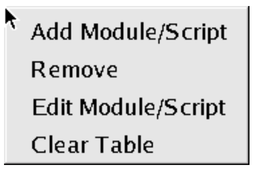
In V6_3_1 software, an **External Synthesis** tab has been added when **zFAST** is the selected synthesizer of an RTL group. This tab shows a table where the modules for external synthesis are listed with their associated Tcl scripts:



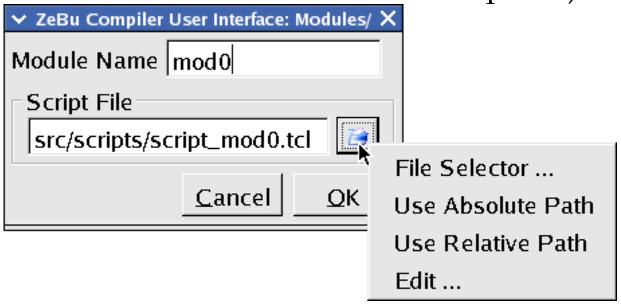
	Module	Script File
1	mod0	src/scripts/script_mod0.tcl

For the modules listed for external synthesis, the parameters for elaboration are extracted by **zFAST** elaboration and an elaborated module is created as an instantiation of the source module with its parameters. The Tcl script should use this elaborated module as an input file for the synthesizer.

A right click in the table opens a contextual menu to create/edit or remove a module/script line:



When selecting **Add Module/Script** or **Edit Module/Script** items, a dialog is displayed with two fields for the module name and the path to the script (an additional menu is available for the selection of the script file):



The Tcl script associated with a module can be a synthesis script or a command-line call to the synthesizer as per user environment.
It is user responsibility that all the necessary information is provided to proceed with the synthesis of the module and that the script generates the proper EDIF file name and path with the proper module name.

The following variables are computed by the ZeBu compiler and are available for use in the Tcl script:

`ORIGINAL_MODULE_NAME`	name of the module in RTL source file
`ELABORATED_MODULE_NAME`	name of the module after elaboration
`ELABORATED_SOURCE_FILE`	name of the elaborated source file
`OUTPUT_EDIF_FILE`	name of the output EDIF file
`OUTPUT_LOG_FILE`	name of the output log file

Note that hierarchical references and accessibility features supported by **`zFAST`** synthesizer, such as RTL paths declaration and CSA, are not available for the modules declared for external synthesis.

3.3.5 New `zFAST` Attribute for memory modeling

In order to optimize the amount of logic for a bit-enabled memory inferred with **`zMem`**, the synthesizer can detect if the read/write in the RTL source files use a fixed sub-word length and map the memory with this sub-word access capability.

For that purpose, the following attribute should be added in the **Additional zFAST Attribute File**:

```
Compile:DetectWordLength=true
```

3.4 RTL-based compilation scripts

3.4.1 New options for the `force_dyn` command

When declaring a signal for runtime control with the `force_dyn` command, it is now possible to connect directly a port declared in the DVE file in order to avoid using the specific C++/C API. The following options can be added for the `force_dyn` command in the RTL-based compilation script:

- To drive the signal itself: `-dve_source <dve_port>` option
- To control the enable signal: `-dve_enable <dve_port>` option

```
force_dyn <signal> -dve_source <dve_port> -dve_enable <dve_port>
```

With this option, `<signal>` cannot be controlled from the C++/C API: only control by driving the connected DVE ports is possible.

3.4.2 Conditional `force assign` command

When the RTL source files have some non-driven nets or ports, the synthesis connects them by default to either GND or VCC. When user wants to control these nets or ports with a register (`force` command with `-value reg` option), the following issues may exist:

- Using the `force undriven` command cannot be used because the nets or ports are actually driven (by GND or VCC) in the netlist resulting from synthesis.
- Using the `force assign` command with pattern-matching declaration (`–fnmatch` option) to control only these nets or ports may cause some inappropriate disconnections of other nets/ports.

It is now possible to declare a condition in the `force assign` command in order to modify only the nets or ports (pattern-matching declaration) which are driven by VCC or GND:

```
force assign -fnmatch –net <net_pattern> –value reg –only_if <source>
[-reg_init <init_value>]
force assign -fnmatch –pin <port_pattern> -value reg –only_if <source>
[-reg_init <init_value>]
```

Where `<source>` is either `"VCC"` or `"GND"` or `"VCC GND"` (not case-sensitive string, with mandatory double quotes) and `<init_value>` can be either `0` or `1`.

Note that `-reginit` is an optional parameter because it is automatically set to `1` if `<source>="VCC"` and automatically set to `0` if `<source>="GND"`).

The `–only_if` option cannot be used for an explicit list of nets/ports.

If some nets/ports in the command are actually driven by something else than a VCC or a GND primitive, these nets/ports are not processed by the `force assign` command and a warning is displayed.

3.5 Back-End Compilation

3.5.1 Hierarchical references in the DVE file

The DVE file now provides a better support of declaration with hierarchical paths:

- assign statements in the DVE file can now be used to force internal signals of the DUT by declaring the hierarchical path of the signal in the left-side of the declaration:
  ```
  assign <path_to_signal_or_vector> = <signal_j>
  ```
  Where `<signal_j>` can be a scalar, a vector or a list of scalars/vectors, as described in the *ZeBu Reference Manual* (Section 2.7 in Rev. B).

- It is now possible to declare an internal signal of the DUT as primary controlled clock (connection to a `zceiClockPort`), as a synthesized clock (connection to a `zClockPort`) or as a Smart Z-ICE/Direct ICE input clock (connection to a `zIceClockPort`).

All other hierarchical references in the DVE file are processed as they were in the previous software releases.

3.5.2 New option to process keeper tristate signals

In the previous releases, when a tristate signal was set to keeper, either through the default resolution (`tristate_default` command) or through individual resolution (`tristate` command), the resolution sometimes led to values which seemed non-deterministic on such signals.

It is now possible to choose between several modes for tristate signals with keeper resolution, in particular to have a deterministic value:

```
tristate_default –resolution <my_resolution> -keeper_mode=<mode> [-
conflict <def_conflict>]
```

Where the possible values for `<mode>` are:

- `sync0`: performance oriented synchronous mode with runtime limitations (for example the `$zClockFilterSel` parameter for glitch filtering cannot be changed to `userClkMask` in the `designFeatures` file). This is the default mode.
- `sync1`: no limitation for runtime features, but with lower runtime performance than `sync0` mode.
- `async`: improved runtime performance but might have non-deterministic behavior. This was the default mode in the previous software releases.

Note that the `-keeper_mode` option can be added with any default resolution (`<my_resolution>` can be `pulldown`, `pullup` or `keeper`).

Other options for the `tristate_default` command (which should be added in the additional command file in the **Advanced → Build System Parameters** frame) are described in the *ZeBu-Server Compilation Manual* (Section 3.7.4 in Rev. C).

3.5.3 Automatic Generation of zCores

When **Automatic Generation of zCores (system-level compilation)** is selected in the **Clustering** panel, it is possible for the user to declare constraints for the compiler through `defcore` commands in a **Manual Declaration of zCores** (**System-level Files** panel). The compiler uses these commands as constraints when processing the zCore generation.

However it is not possible to declare overlapping zCores as user constraints with automatic zCore generation, as shown in the following example (precisely because `A.B.X` belongs to `A.B`):

```
defcore coreX -pathlist={A.B.X}
defcore coreY -pathlist={A.B}
```

3.5.4 Static Timing analysis new features

- Width and depth information for DDR2- and RLDRAM-based memories is now included in the reports.
- In the reports, the physical and logical IDs of FPGAs are now displayed differently:
  - Physical FPGAs: `Ux/My/Fz`
  - Logical FPGAs: `Ux_My_Fz`
- Clock paths going through clock dividers are now taken into account.
- The primary clocks generated by the ZeBu clock server are now reported as IF→F*n* in the static timing analysis reports, in the same way as other inter-FPGA clocks.

3.5.5 Timing-driven mode for System Place and Route

- The results obtained with the **Timing-driven Mode** have been improved by 10% to 20% with respect to the previous release.
- When the **Timing-driven Mode** is active, the following information can be found in the log of the **Place and Route System** step for each iteration:
  - A reference of the iteration number (`<n>`):
    ```
    # step zParTiming : Iteration <n>
    ```
  - Estimated value runtime frequency and clock skew resulting from Static Timing Analysis.

Note that the actual timing estimations may be different from the one displayed at the end of the **Place and Route System** step. The information intended for runtime (and automatically set if the **Save timing Parameters for Runtime** option is selected in **`zCui`**) is the one displayed in the log of **Create Timing DB** step in **`zCui`**.

3.5.6 Memory Modeling

3.5.6.1 Controlling the maximum number of ports for zrm-based memories

By default, the zrm-based memory models support up to 8 ports. It is now possible to modify the maximum number of ports for zrm-based memory models according to the type of physical memory resources:

```
configure_memory RLDRAM|DRAM -max_port <nb_port>
```

This command for the zCore-level compiler should be declared as described in the *ZeBu-Server Compilation Manual* (Section 5.5.1 in Rev. C).

This modification allows in particular more ports on RLDRAM-based memory models than on DDR2-based memory models in order to improve the runtime performance and to support more memory models with density-oriented clustering.

Example:
To allow a maximum of 16 ports for RLDRAM memory banks:

```
configure_memory RLDRAM -max_port 16
```

3.5.6.2 Transaction-based Communication for DDR2 memories

When the achievable runtime frequency (`driverClock` frequency) is constrained by DDR2-based memory models (max frequency 2-3 MHz), the communication with the DDR2 memories can be changed to a transaction-based mode. The achievable runtime frequency may increase to ~20 MHz in most cases but every read-access to a DDR2-based memory will be longer than in non-transactional mode.

The transaction-based communication can be activated by adding the following command in the additional command file in the **Advanced → Build System Parameters** frame:

```
enable zrm_transactional_mode
```

Note:
When the `driverClock` frequency is constrained by zrm memory accesses, the log of the **Create Timing DB** step in **`zCui`** is similar to the following example:

```
#   step REPORT :
#--------------------------------------------------------------------------------#
#   step REPORT :   No inter-fpga asynchronous set/reset path found
#   step REPORT :   No inter-fpga filter path found
#   step REPORT :   Longest inter-fpga clock path delay is : 25 ns
#   step REPORT :   Max pessimistic delay (clock skew + routing data) is: 99 ns
#   step REPORT :   Critical routing data path delay : 74 ns
#   step REPORT :   . Constant part    : 41 ns
#   step REPORT :   . Multiplexed part : 33 ns
#   step REPORT :   Xclock frequency is : 400 MHz
#   step REPORT :   Longest memory period is : 341 ns
#   step REPORT :   No flexible probes found
#   step REPORT :   Driver clock frequency is limited by memories
#   step REPORT :   The theoretical frequency using default settings and
ignoring clock skew is 2925 Khz
#   step REPORT :
#--------------------------------------------------------------------------------#
```

3.5.6.3 Memory Import

The Memory Import feature which consists in creating a logical view of a memory instance after compilation has been improved. It is now possible to use this feature for memories created from **zMem** memory models, in particular to provide a logical view which is different of the physical memory model.
As in previous software versions, this feature is based on the `import_mem` command of **zDbPostProc** and a memory import script, as described in the dedicated *ZeBu Application Note* (AN021, Rev. A). In the memory import script, the size of memories can now be declared with binary units (k, K or M).

3.6 Runtime database

3.6.1 New options to launch zDbPostProc

It is now possible to load the runtime database faster when launching **zDbPostProc** or **zSelectProbes**:

- By default, the database is opened in read-only mode supporting the usual selection operations. When nets have several identifiers in the database, these are available (when the `print_equi` command is used or when the **aliases** button is clicked in **zSelectProbes**).
- To load the entire database when launching **zDbPostProc** or **zSelectProbes**, in particular to optimize the duration of bulk operation on the database such as vectorization, it is possible to launch the tools with the following option:
  ```
  $ zDbPostProc –full [-rw]
  ```
  or
  ```
  $ zSelectProbes –full [-rw]
  ```
- To speed-up the load time, information about relationship between the nets to list aliases can be completely ignored:
  ```
  $ zDbPostProc –noequi [-rw]
  ```
  or
  ```
  $ zSelectProbes –noequi [-rw]
  ```
  Note that the `print_equi` command and the **aliases** button are not disabled but they show `No aliases`, whatever the actual content of the database.
- The `-rw` option can be used alone or with `–full` or `–noequi` for advanced operations on the database, such as vectorization or memory import.

3.6.2 New selection command for off-line CSA in zDbPostProc

In order to select all the registers for a proper support of the off-line CSA feature, the following command can be used in the **zDbPostProc** script:
```
select_all_csa_support_regs
```
There is no equivalent button in **zSelectProbes** interface. Equivalent selection can be done with the simultaneous selection of NON-CSA, DUT RTL, DUT EDIF, and ZEBU SYN.

3.7 Runtime

3.7.1 zRun Tcl command field

In addition to the commands corresponding to the GUI operations, the `zRun_cmds.tcl` file generated by **zRun** now includes all the `ZEBU_xxx` functions which are launched from the **Tcl Command** field of **zRun**. Any `source <myScript.tcl>` command is also stored in the `zRun_cmds.tcl` file. Only the commands executed successfully are actually stored.

Note that executing clock control functions such as `ZEBU_Clock_disable`, `ZEBU_Clock_enableForever`, and `ZEBU_Clock_enable` from the Tcl Command field through a source `<myScript.tcl>` command, the **Monitor** window and the clock group panels in the **Run Control** window are updated automatically.

3.7.2 Temperature log file

A new temperature log file, `temperature_<yyy_mm_dd>.log`, is generated in the `$ZEBU_SYSTEM_DIR/logs/temperature` directory and includes three types of information:

- System events (per system)
- Temperatures (per module)
- Fan speeds (per unit)

See more details in *ZeBu-Server Installation Manual* (Section 7.7 in Rev. F).

3.7.3 New C++ method / C function to get the driverClock frequency

This release introduces a new method/function in the C and C++ APIs to get the `driverClock` frequency (given in kHz), which is equivalent to the `ZEBU_getDriverClkFrequency` Tcl command of **zRun**.
This method/function should be called after the connection to the ZeBu system (`Board::Open` in C++ / `ZEBU_Board_Open` in C)

In the C++ API:
```
void Board::getDriverClockFrequency(unsigned int&frequency);
```

In the C API:
```
void ZEBU_Board_getDriverClockFrequency(ZEBU_Board *board, unsigned int
*frequency);
```

3.7.4 Memory content initialization from RTL source files

When the RTL source files of the design include memory initialization commands (`readmemb/readmemh`), an initialization file (`readmem.dump`) is generated by **zCui** when synthesizing with **zFAST**. This file is now automatically loaded at runtime to initialize the memories of the design when connecting to ZeBu.

However, explicit initialization from the `designFeatures` file has priority: if some memories initialized in the `readmem.dump` file were declared by user for initialization in the `designFeatures` file, the RTL initialization will not be taken into account.

3.7.5 Other runtime new features

This release introduces the following enhancements for runtime:

- It is now possible to launch **zInstall** while a test is running. **zInstall** is queued as any other runtime task.
- In a multi-user environment, the user identifier is now allocated automatically when launching emulation.
  Manual declaration with the `ZEBU_SRD_SET_RUNID` environment variable is no longer necessary and may cause some conflicts.
- For debugging purposes, it is now possible to display the `pstack` result and/or the `gdb` result in case of segmentation fault or in case of timeout of a read access to the ZeBu system. Using `pstack` is useful in particular for multi-thread environments in order to get information on the failing thread and of other threads which may have caused the error.
  These features can be activated by setting the following environment variables:
  ```
  $ export ZEBU_DEBUG_STACK_USE_PSTACK=OK
  ```
  and/or
  ```
  $ export ZEBU_DEBUG_STACK_USE_GDB=OK
  ```
- A function has been added in the ZeBu C API to get the EDIF instance path from an RTL instance path, as requested in RT#27206:
  ```
  const char *esx_getEdifPathOfIns (const char *name);
  ```
  This method is described in `libzRtl_C.h` header file. The corresponding C++ method already existed.
- The memory reserved for the reception FIFO of messages for the transactors is shared by all the ports. In order to provide information to the user about memory usage, the `designFeatures.help` file now includes information about the number of messages actually used for each port:
  ```
  ###############################################
  ################# xTors PORTS ##############
  ###############################################
  ######## xTors features declaration ######
  $xtor_0::zceimessageoutport_0.nbMessMax = 4095;
  ```
  Moreover, the error message has changed to:
  ```
  -- ZeBu: tb_dut : ERROR : LUI1848E : Cannot initialize port
      "xtor_3::zceimessageoutport_0", memory is not reserved for port 1.
  -- ZeBu: tb_dut : ERROR :          : You should decrease the size of the FIFO used by
     all the txp ports, for example by adding
     "$xtor_3::zceimessageoutport_0.nbMessMax = #nbMessMax;" in your
     designFeatures file, where #nbMessMax should be less than the current value (62).
  -- ZeBu: tb_dut : ERROR : : The current values are :
  -- ZeBu: tb_dut : ERROR : : xtor_0::zceimessageoutport_0.nbMessMax = 4095
  -- ZeBu: tb_dut : ERROR : : xtor_0::zceimessageoutport_1.nbMessMax = 2047
  -- ZeBu: tb_dut : ERROR : : xtor_1::zceimessageoutport_0.nbMessMax = 2729
  -- ZeBu: tb_dut : ERROR : : xtor_1::zceimessageoutport_1.nbMessMax = 2047
  -- ZeBu: tb_dut : ERROR : : xtor_2::zceimessageoutport_0.nbMessMax = 909
  -- ZeBu: tb_dut : ERROR : : xtor_2::zceimessageoutport_1.nbMessMax = 2047
  -- ZeBu: tb_dut : ERROR : : xtor_3::zceimessageoutport_0.nbMessMax = 247
  -- ZeBu: tb_dut : ERROR : : xtor_3::zceimessageoutport_1.nbMessMax = 2047
  ```

3.8 ZEMI-3

New C++ methods in the ZEMI-3 API are available to activate and control the WaitGroup capability, in replacement of the deprecated environment variables (see Section 2.9.2). These methods are described in the `ZEMI3Xtor.hh` header files (which must be included in the source files of the testbench).

The `ZEMI3Xtor::UseWaitGroup` method turns on or off the WaitGroup capability for Rx ports of the transactors, with a default timeout of 1000:

```
ZEBU::ZEMI3Xtor::UseWaitGroup (bool on, int timeout=1000);
```

The `ZEMI3Xtor::UseYield` method turns on or off the use of a yield mechanism to break message ports polling loops for both Rx and Tx ports of the transactor in the header of the method, `rx` and `tx` are testbench-centric which means `rx` is for HW → SW messages):

```
ZEBU::ZEMI3Xtor::UseYield (bool rx, bool tx);
```

These methods should be used in the testbench, and are suitable for System C environments with `ZEMI3Manager` class, or the zcei/ZEMI-3 mixed environments.

When using **`zEmiRun`**, there are command line options which are equivalent to these methods:

- `-w|-waitgroup <timeout>`: Use WaitGroup with the specified timeout (in microseconds). Default is no WaitGroup.
- -`y|-yield <string>`: Yield in Rx/Tx loops. Possible values for `<string>` are `rx`, `tx`, `all` or `off`. Default is `off`.

3.9 SystemVerilog Assertions

In V6_3_1 software version, the SVA support when synthesizing with **`zFAST`** has been improved for both Full Report and Report only Failure modes.

In particular, combinational SVAs, coverage functions and concurrent assertions are now supported, but with specific limitations for Full Report mode (listed in Section 5.8.2) and for Report only Failure mode (listed in Section 5.8.3).

The SVA features that were introduced in the SystemVerilog 2009 standard can be enabled by setting the following attribute in the **Additional zFAST Attribute File**:

```
hcsrc:sv2k9=true
```

The dedicated *ZeBu Application Note* (AN028) has been updated in the document package delivered with V6_3_1 software release.

3.10 Synchronous Clock Handling with Smart Z-ICE

When the Smart Z-ICE interface provides a primary clock to the design, for example when connecting a JTAG debugger, this clock is asynchronous to the clocks generated by the ZeBu clock generator.
Such an asynchronous clock cannot be processed with the glitch filtering feature described in the *ZeBu-Server Compilation Manual* and may cause a wrong runtime

behavior, in particular unexpected waveforms when tracing with the `SRAM_TRACE` driver.
To avoid this, the ZeBu compiler can use a fast primary controlled clock (output of a **`zceiClockPort`**) to sample the Smart Z-ICE input clock. For that purpose, some specific declarations are necessary in the DVE file, as described in the ***Zebu-Server Smart Z-ICE Manual*** (Section 3.5.3 in Rev. D).

3.11 Optimizing Runtime Performance

The features described in this section are applicable for optimization of runtime performance and should not be used when bringing up the design.

3.11.1 Timing Optimization for Automatic Generation of zCores and Performance Oriented Partitioning

When working to optimize the runtime performance, it is now possible to activate **Timing Optimization** for automatic zCore generation and/or for performance oriented partitioning. There are dedicated checkboxes (**Timing Optimization**) in the **Automatic Generation of zCores (system-level compilation)** and **Performance Oriented Partitioning (zCore-level compilation)** frames of the **Clustering** panel.
Note that timing optimization makes the compilation process longer and should thus be used with much care.

Once activated, the **Timing Optimization** frame offers two different choices:

- **Post-partitioning optimization**:
  With this mode, the design is first partitioned to optimize the cut between zCores (or between FPGAs) and then the longest combinational paths are analyzed to choose a solution which minimizes the hops between zCores or between FPGAs.
  When this mode is activated, the automatic zCore generation/partitioning lasts typically ~2x as much as without timing optimization.
  This mode is recommended when the long combinational paths in the design are quite short (3 or 4 hops).
- **Timing driven partitioning**:
  With this mode, the analysis for longest combinational paths is done before optimizing the cut between zCores or between FPGAs.
  When this mode is activated, the automatic zCore generation/partitioning lasts typically ~5x as much as without timing optimization.
  This mode is recommended when the long combinational paths in the design are long (~10 hops). However, it may strongly impact the cut in case of a high number of long paths.

3.11.2 Optional optimizations for flexible probes

By default, the processing of flexible probes by the compiler optimizes the amount of logic resources used. However, when targeting runtime performance, a new option can be added when declaring flexible probes in an RTL-based compilation script:

```
probe_signals -type flexible [-optimize speed] [other options]
```

Note that this optimization uses a higher amount of logic resources and is not efficient in the following cases:

- Many flexible probes per FPGA
- Many groups of flexible probes per FPGA
- Simultaneous use of DPI calls and/or System Verilog Assertions.

3.11.3 Investigating and removing inter-zCore combinational paths

When working on improving the runtime performance of in a design, the number of combinational, looped interconnections between zCores (asynchronous paths going through several zCores) should be reduced.

To display information about these inter-zCore combinational paths in the log file (**Build System** step), the following command should be added in the additional command file in the **Advanced** → **Build System Parameters** frame:

```
winding_path stats [depth_threshold=<integer>]
```

With this command, the netlist is not modified (no reduction of the inter-zCore combinational paths) and the compilation process is continued.

It is possible to replicate some logic in several zCores in order to improve the performance (reducing the number of interconnection between zCores in most cases). For that purpose, the following command should be added in the additional command file in the **Advanced** → **Build System Parameters** frame:

```
winding_path duplicate_logic [depth_threshold=<integer>]
```

Details about the `depth_threshold` option and examples will be added in the next revision of the *ZeBu-Server Compilation Manual* (Rev. D).

3.11.4 New Option for Clock Modeling

When working on reducing the clock skew, the **Glitch Filtering** option in the **Back-End** → **Clock Handling** panel may cause runtime issues and the impact of **Clock Localization in a Single FPGA** option is severe for FPGA filling rates and runtime accessibility, as described in Section 5.5.2.

It is now possible to improve the clock skew reduction by adding the following command in the additional command file in the **Advanced** → **Build System Parameters** frame:

```
clock_handling filter_glitches_synchronous –FGS_fetch_mode=yes
```

Note that using this command disables any setting in the **Clock Handling** panel of **`zCui`**. If other options than **Glitch Filtering** were set in **`zCui`**, they are overridden by the above command. However, it is possible to manually add other options to the `clock_handling` command to activate the expected features.

In some cases, the activation of this mode in replacement of clock localization is not possible, in particular if memories are present in the clock tree or if both clock edges are active on a design clock which would be as fast as `driverClock/2`.

3.11.5 Runtime advanced setting to avoid clock race issues

Increasing the `driverClk` frequency may raise some clock race issues on critical paths in the design. To avoid such issues, it is now possible to declare for runtime a specific delay between subsequent edges of the design clocks, assuming that they are controlled clocks declared in the same clock group.

This declaration can be done in the `designFeatures` file, using the `delayClkEdge` parameter. Up to 16 different constraints can be declared in the `designFeatures` file. The delay is given as a minimum number of cycles of `driverClk`.

The following syntax is for edges of two clocks, which is the most frequent usage:

```
$delayClkEdge_<x> = "NbDriverClk <N> FROM{<edge_i>(<clk_i>)}
TO{<edge_m>(<clk_m>)}";
```

Where:

- `<x>` is an integer identifier of the constraint, ranging from 0 to 15.
- The `FROM` indicator is followed by the first clock edge (placed between curly brackets) which is considered to set the constraint. The `TO` indicator is followed by the second clock edge which is considered to set the constraint (placed between curly brackets).
- `<N>` is the minimum number of `driverClk` cycles to occur between the subsequent edges of 2 design clocks, ranging from 1 (normal mode) to 31.
- `<clk_i>/<clk_m>` are the names of the controlled clocks.
- `<edge_i>/<edge_m>` indicates on which clock edge the delay is applicable. It is one of the following values (not case sensitive): `RISE`, `FALL` or `BOTH.`

Notes:

- If you want to declare such constraints for all the controlled clocks of the design (assuming they are in a single group), you can use the following syntax:
  ```
  $delayClkEdge_<x>="NbDriverClk <n> FROM{ RISE(*) } TO{ RISE(*)}";
  ```
- For the FROM and/or the TO indicator, it is possible to declare several clock edges by declaring a list of edges and clock signals for which the constraint will be set. The | separator is used in the list:
  ```
  FROM { (<edge_i> <clk_i>) | (<edge_j> <clk_j>) | ...}
  ```

4 Documentation Package

4.1 Master document in the documentation package

The ZeBu documentation package includes a master document which is a PDF file with bookmarks for direct access to all documents of the package (in the bookmark panel of Acrobat).

Each document of the package includes a specific bookmark for the master document (available at the top of the bookmark tab of Acrobat for this Release Note).

For ergonomics purposes, the master document includes a bookmark which opens the **Acrobat Full Search** window.

4.2 ZeBu Manuals

The following table lists the complete documentation package, showing the manuals which are available for ZeBu-Server and mentioning which manuals can be used when ZeBu-Server dedicated manuals or up-to-date ZeBu common manuals are not available. Note that new and updated documents are in *italics*.

Ind.	Manual Name	Rev	Comments
01	***Installation Manual***	*f* ***ZSE***	*Major update for V6_3_1.* *See detailed History table in the manual.*
02	**Compilation Manual**	c **ZSE**	Last update for V6_3_0.
03	**HDL Co-Simulation Manual**	b **ZeBu**	Last update for V 3_1_0.
04	**C++ Co-Simulation Manual**	b **ZeBu**	Last update for V4_3_0.
08	**zRun Manual**	d **ZeBu**	Major update for V6_3_0. See detailed History table in the manual.
10	***Smart Z-ICE Manual***	*c* ***ZSE***	*New section:* • *Declaration for synchronous clock handling with Smart Z-ICE.*
11	**Direct ICE Manual**	c	Minor update for V6_3_0.
12	***ZeBu Reference Manual***	*b*	*Major update for V6_3_0.* *See detailed History table in the manual.*
13	***C API Reference Manual*** ***C++ API Reference Manual***	*V6_3_1*	
14	**ZEMI-3 Manual**	b **ZeBu**	Last update for V4_3_3.
15	**zFAST Manual**	a	First Edition for V6_2_1.

4.3 Additional documents

In order to keep track of recent new features not yet described in the ZeBu Manuals, Release Notes for previous releases are available in the documentation package.

The following Application Notes are included in the documentation package and can be used with the present release. Note that new and updated documents are in *italics*.

Ind.	Application Note	Rev	Comments
AN017	***Timing Analysis with `zTime`***	*c*	*Updated for V6_3_1.*
AN021	**Importing Memories for Easy Runtime Access**	**a**	
AN022	**`ZW-FPGA` Usage**	**d**	Added support of **`zFAST`** synthesizer and new components for ZW-FPGA V1.2.
AN025	**Integrating with ZeBu thread-safe library**	**b**	
AN028	*SystemVerilog Assertion (SVA) for ZeBu*	*b*	*Major update for V6_3_1.* *See detailed History table in the document.*
AN029	**`zDPI` Feature for ZeBu**	**a**	
AN032	**Migrating from ZeBu-XXL to ZeBu-Server**	**a**	

For your reference, the license agreement for the ZeBu software and for usage of the third-party software packages delivered with the ZeBu software is available as a PDF file in the documentation package.

- The Xilinx license agreement is included.
- The Concept Engineering license agreement is also available for information about RTLvision PRO and GateVision PRO usage.

5 Known Limitations

This chapter lists the known limitations in the V6_3_1 release.

5.1 Operating Systems

5.1.1 Specific SUSE Linux Enterprise Server 10 requirement

The present version of the ZeBu software requires the availability of the `compat-readline-4.3` package which is not installed by default on SUSE Linux Enterprise Server 10 operating system.

5.1.2 Red Hat Enterprise Linux 5

The present version of the ZeBu software can be used with Red Hat Enterprise Linux 5 with the following limitations:

- The `autofs` Linux functionality, which allows automatic mounting of a remote disk volume, does not work correctly (note that this Red Hat Enterprise Linux 5 limitation is unrelated to ZeBu software).
  It is necessary to install the following RPM package managers for Red Hat Enterprise Linux 5:
    - `kernel-2.6.18-53.1.6.el5.x86_64.rpm`
    - `kernel-devel-2.6.18-53.1.6.el5.x86_64.rpm`
    - `kernel-headers-2.6.18-53.1.6.el5.x86_64.rpm`

Some additional recommendations must be taken into account:

- You have to compile any software which is linked to a ZeBu API (binary library) such as a C/C++ testbench or the software part of a transactor using `gcc/g++` version 4.1. This is the default version for this operating system.
- The C++ standard library version 6 (`libstdc++.so.6`) is the default version for this operating system.
- The Linux environment variable `LD_ASSUME_KERNEL` must be unset.

5.1.3 GTKWave limitation

The package for GTKWave waveform viewer can be installed only with Red Hat Enterprise Linux 4 operating systems, but not with Red Hat Enterprise Linux 5 or SUSE Linux Enterprise Server 10.

Once installed from a RHEL4 workstation, the program runs correctly on all supported platforms.

5.2 No communication with units when a PC is OFF

In a multi-user system configuration (either single- or multi-unit), if one of the host PCs connected to any unit is switched OFF, you may no longer be able to communicate with the units from any PC connected to the system. In such a case, the following error will be thrown:

```
-- ZeBu :  zUtils : Loading U0_BP_FC0 with "/usr/share/zebu/V6_3_1/etc/firmwares/ZSE/
           default/zse_sbp_2s_fc.bit" (using CPU).
.......... LAU0331E : Cannot check if U0_BP_FC0 is done (cannot read status).
-- ZeBu : zUtils : ERROR : FAILED.

-- ZeBu : zUtils : WARNING : Retrying to load U0_BP_FC0 (1/10).
-- ZeBu : zUtils : Loading U0_BP_FC0 with
          "/usr/share/zebu/V6_3_1/etc/firmwares/ZSE/default/zse_sbp_2s_fc.bit" (using CPU).
          LAU0330E : Cannot reset U0_BP_FC0 (cannot check cmd 0x00000002).
-- ZeBu : zUtils : ERROR : FAILED.
```

In most cases, communication resumes as soon as ALL the PCs are switched ON. If not, you must power OFF/ON all units then launch either **`zSetupSystem`** or **`zUtils`** `-initSystem`. For more details, please refer to ***ZeBu-Server Installation Manual***.

5.3 Compilation interface (`zCui`)

5.3.1 `zCui` uses `/bin/sh` and ignores `.cshrc` aliases/functions

`zCui` uses `/bin/sh` and ignores aliases and functions defined elsewhere (for example in `.cshrc`). Make sure you use scripts in your PATH instead of aliases.

5.3.2 Limitations with multiple RTL Groups

In **`zCui`**, the RTL source files are gathered in one or several RTL Groups. By default, **`zCui`** creates one RTL group (**Default_RTL_Group**).

It is recommended to have only one RTL group for the project, since most of the ZeBu debugging features are supported for one group only, in particular SystemVerilog Assertions, zDPI feature, RTL-based compilation scripts and Combinational Signals Accessibility (CSA). Note that hierarchical references in the RTL sources are supported only inside an RTL group (no hierarchical references between different RTL groups can be synthesized).

Additional RTL groups can be of interest when integrating RTL sub-projects or IPs, when investigating synthesis issues or when a sub-module needs some specific synthesis features for performance purposes in particular.

Each RTL group has its own synthesis options and synthesis output files are automatically merged by the ZeBu compiler.

The declaration of RTL source files is described in detail in the ***Zebu-Server Compilation Manual***.

5.3.3 Limitations in the Compilation view

- The status icon is not displayed for the groups of tasks.
- The comment for the **FPGA Place & Route** branch may indicate a wrong value. For example, if a single task passes then the following value could be reported even if some tasks failed:

```
1p, 0f/2 tasks
```

- The **`zCui`** task engine may block when an FPGA actually succeeded but **`zCui`** never received the `SIGCHLD` signal from the operating system. In such a case, the compilation does not finish correctly and must be manually aborted by user. This FPGA will be displayed as failed.
- At the end of each compilation, the winner FPGA parameters are stored in a file located in the Back-End directory. However this file is re-written after each compilation, leading to the loss of previous winner FPGA parameters after each compilation.

5.3.4 Other limitations

- The **Merge Case** checkbox has been removed from the **Debug Database Parameters** frame in the **Advanced** panel because it matched a deprecated command of **`zDbPostProc`**.
- Debug options are not gathered in a single panel.
- When the **`zCui`** archiver feature is used in graphical mode, only one back-end can be selected at a time.
- The **Post-Compilation FPGA Summary** and **Interactive FPGA Summary** are not available. The corresponding icons and menu items have been removed.
- The information displayed in the **FPGA P&R** panel corresponds to the original Place & Route settings but it is not updated with the actual parameter set which was used to pass the Place & Route.

5.4 zFAST Synthesis

5.4.1 Functional limitations

- Top-Down Synthesis:
    - The generated netlist is flattened or partially flattened. That can introduce clustering issues since clustering is based on hierarchy.
    - Hierarchical references are not supported.
- When there are several RTL groups in the design, the declaration of a top name for each group is mandatory. If there is only one RTL group, the name of the top of the design will be used for the group.
  For other limitations with multiple RTL groups, see Section 5.3.2.
- The top of the design cannot have a Verilog escaped identifier.
- When a used file from a library directory or from an include directory is deleted, **zCui** reports an error instead of proceeding with elaboration.
- For a VHDL design, the memories inferred by **zFAST** using the ZeBu memory generator sometimes result into inappropriate memories (in particular for a too high number of ports).
  When facing this issue, the following line should be added in the **Additional zFAST Attribute File**:
  ```
  (onset zFe:App (if (= (get zFe:App) hcs_mixed)
     (set compile:detectmemory false)))
  ```

5.4.2 Interface limitations with zCui

When selecting **Verilog95** in the **Main** → **Verilog Language** list, **zFAST** actually synthesizes the design with Verilog2001 constraints, in particular the Verilog2001 keyword usage in Verilog95 code will cause synthesis errors.

5.4.3 Synthesizer inefficient when a variable is used in many places

The synthesizer is inefficient when a variable of a large array or structure type is used in many places (hundreds or thousands after loop unrolling).

5.5 Compilation

5.5.1 Limitations for declarations with RTL paths

- For a VHDL design or a mixed-language design with VHDL, RTL names are not fully supported for the back-end compilation or runtime access, in particular for CSA (Combinational Signals Accessibility) feature.
- Curly brackets ("{" and "}") are usually not supported by synthesizers. If some are present in the RTL paths in RTL-based compilation scripts, a workaround is to use the EDIF path in which support for special characters is better.
- When RTL-based commands are used with pattern matching (`-fnmatch`),

pattern-matching characters ("*", "[" and "]") cannot be present as in RTL paths.

- Mixed RTL/EDIF paths are not supported, for example a mixed path including an RTL path down to the following EDIF macro:
  ```
  probe_signals -wire top.path.rtl.suffix.in.edif.wirename
  ```
  A new error policy will cause **zFAST** to exit in error when a path ends in an EDIF macro.
- The RTL hierarchical path to an unconnected port of a blackbox in the design cannot be used in a RTL-based command.
- zCore declaration with RTL paths is not supported when synthesizing with **zFAST** in top-down mode or with third-party synthesizers. It is only supported when synthesizing with **zFAST** in block-based mode.
- It is not possible to use RTL-based declarations in the DVE file to declare the path to a `generate` block with an escaped ID, as in the following example:
  ```
  top.genblk1.\BLOCK.ESCAPED[0] [0].ins.\OUT.result ,
  ```
  Where `\BLOCK_ESCAPED` is declared as in the following RTL source code example:
  ```
  module bb(CKOUT);
  ...
  module sub (IN, CK, OUT);
  ...
  module top(\IN.0 , \OUT.0 );
  input [5:0] \IN.0 ;
  output [5:0] \OUT.0 ;
  genvar i;
  generate begin
  for (i = 0; i < 2; ++i) begin : \BLOCK.ESCAPED[0]
  wire wckout;
  bb bbck(.CKOUT(wckout));
  sub ins(.IN(\IN.0 [i+2]), .OUT(\OUT.0 [i+2]), .CK(wckout));
  end
  end
  endgenerate
  endmodule
  ```

5.5.2 Compilation settings impacting runtime accessibility

When the following options are set to compile a design, some signals may not be available for runtime accessibility (for both read and/or write access) and thus may impact the CSA feature:

- **Single FPGA clock path localization** option in the **Clock Handling** panel.
- `winding_path` command in the advanced command file in the **Advanced → Build System Parameters** frame.

5.5.3 Clock connection to the `SRAM_TRACE` driver

Connecting a primary clock (`zceiClockPort` output) as a data signal in the instantiation of the `SRAM_TRACE` driver may cause a routing error during FPGA Place & Route.

5.5.4 Limitations for memory modeling

5.5.4.1 Limitation on clocks for DDR2 zrm memories:

Because of an inappropriate frequency range, it is not possible to drive a DDR2 zrm memory instance with a synthesized uncontrolled primary clock generated by the frequency synthesizers (declared by instantiating a `zClockPort` in the DVE file). The minimum frequency of the synthesizers is much too high to connect such a clock to DDR2 memories.

5.5.4.2 Limitation on Read/Write priority for multi-port zrm memories:

The read-before-write priority is set individually on one port. The read/write priority between two ports of the same memory instance is not deterministic for zrm memories, even if both ports use the same clock signal.

5.5.4.3 Multi-port Memories in IF

When a ZEMI-3 or a zcei-based transactor is synthesized with **`zFAST`**, memory inference may result into multi-port LUTRAM-based or BRAM-based memory models with 3 ports or more, which cannot be mapped in the RTB FPGA by the ZeBu compiler.

5.5.5 Automatic zCore Generation and Performance Oriented Partitioning

The resources used by flexible probes, CSA, and zDPI features are not taken into account during automatic zCore generation and performance oriented partitioning (in system-level and zCore-level compilation).

5.5.6 Wildcard characters in Density Oriented Clustering advanced commands

Some zCore-level compiler `cluster` commands supporting wildcard characters may generate undesired effects if they are not explicitly redirected to the right zCore.

Example:

```
zcorebuild_command * {cluster core add name=mycore  top.core_inst.inst2.*}
```

Assuming we have:

```
defcore  CORE0 -path_list {top.core_inst}
defcore  CORE1 -path_list {top}
```

The `cluster core` command will be correctly transmitted to `CORE0`. However, the system-level compiler may leave some logic in other zCores under the same path (`top.core_inst.inst2`).

The solution is to write the commands in a script for the zCore-level compiler, as described in the *ZeBu Compilation Manual* (Section 5.5.1 in Rev. C), instead of adding only `zcorebuild_command`.

5.6 Runtime database

5.6.1 Limitations for SystemC co-simulation wrapper

In this release, it is not possible to generate the SystemC co-simulation wrappers with **`zDbPostProc`**/**`zSelectProbes`**. This limitation is also applicable when the corresponding checkbox is selected in the **Wrapper** tab in **`zCui`**.

5.6.2 Limitation for the hierarchical separator in `zSelectProbes`

If the hierarchical separator of the runtime database has been changed (to "`|`" or "`/`" for example) with the `separator` command of **`zDbPostProc`**/**`zSelectProbes`**, the default separator "." may still prevail in the **`zSelectProbes`** display.

5.6.3 Inappropriate behavior when expecting display of CSA signals

The **CSA** button in the **`zSelectProbes`** interface does not show the simulated signals resulting from on-line CSA as user may expect: it also shows some signals that were processed during CSA but which are not relevant for user.

There is no way to show only the simulated signals in **`zSelectProbes`**.

5.7 Runtime limitations

5.7.1 Initialization order in a C++/C testbench

In a C++/C testbench, for co-simulation or transaction-based emulation, it is mandatory to connect the message ports after connecting to the system with the `Board::open` method / `ZEBU_Board_open` function but before initializing with the `Board::init` method / `ZEBU_Board_init` function.

Note that the methods/functions to connect the message ports differ with the verification environment:

- `Driver::connect` method/`ZEBU_Driver_connect` function for C++/C co-simulation.
- `Port::connect` method/`ZEBU_Port_connect` function for transaction-based environments.

If the order of initialization is not correct, the emulation stops with the following error message (the identifier of the error may be `ZHW0810E` or `ZHW0761E` according to your verification environment):

```
Unable to connect the port <driverName>::<portName>.
You must register the port just between the open and the init functions.
```

5.7.2 Concurrent use of Flexible Probes, SVA and DPI calls

- When the verification environment uses simultaneously flexible probes, SVAs and DPI calls (or at least two of them), the initialization of the clocks for these features may cause unexpected runtime results for these features.
- When using flexible probes and DPI calls simultaneously, it is not possible to select all the flexible probes at once with `".*"` for the group in the `FlexibleLocalProbeFile::add` method. You have to declare explicitly the name of the group when calling the `FlexibleLocalProbeFile::add` method.

5.7.3 Deprecated `run` method in non-blocking mode with C++ co-simulation

The *ZeBu C++ Co-Simulation Manual* (Section 6.2.4 in Rev. B) describes the execution of cycles using the `CDriver::run` method in non-blocking mode:

```
unsigned int CDriver::run (unsigned int numCycles, bool block = false)
```

The `block` parameter is now ignored by the ZeBu runtime software. Therefore the behavior of the `run` method is always in blocking mode. If for some reason (for ex. a trigger event) the specified `numCycles` cannot be executed, the C++ testbench will hang in the `run` method.

To get around this inconvenience, use the `CDriver::wait` method instead:

```
unsigned int CDriver::wait(unsigned int trigger_mask, unsigned int
numCycles=0xffffffff);
```

Where each of the 16 `trigger_mask` LSBs must be set to `1` to stop the run if the corresponding trigger value is `1`. If no trigger is fired, then the specified number of cycles will be executed.

5.7.4 Unsupported waveform file formats (`.bin` and `.vpd`)

The binary proprietary format (`.bin` waveform files) and the vpd format are not supported with the present software release.

5.7.5 Restrictions when dumping wavefiles using the `WafeFile` class

Wavefiles can be generated for internal signals synchronized with a controlled clock or with `driverClk` by creating an object in the `WaveFile` class.

Signals can be added or removed from the wavefile as required, without affecting the DUT and the hardware part of transactors. The resulting waveform file contains all dynamic probes selected via **`zSelectProbes`** and **`zDbPostProc`** (see *Accessing Dynamic Probes from the C++ Testbench* in the *ZeBu C++ Co-Simulation Manual*) or a set of dynamic probes explicitly added into the wavefile at runtime. The drawback is the cost in speed. However, this feature can be enabled/disabled (`dumpon/ dumpoff` methods) to dump only during a simulation time-window where the data is critical for debugging.

However, using the `WaveFile` class imposes the following restrictions:

- This class is only supported with the threadsafe versions of `libZebu.so`: `libZebuThreadSafe.so` and `libZebuThreadSafeDebug.so`.

- **zRun** is not supported while a file is open.
- You can only open one file at a time.
- You cannot access signals, memories and clocks while a file is open.
- You cannot save the hardware state (with `saveHardwareState` method), if the `dumpoff` method was not called before. Else, a deadlock will occur in the `saveHardwareState` method.
- You must close or destroy the file before closing ZeBu, else a deadlock occurs.

`WaveFile.hh` header file describes the `WaveFile` class. Its interface is similar to the `Board` and `Driver` classes used to dump interface and internal signals in C++ co-simulation.

5.7.6 Disk resource conflicts for `.ztdb` files

The `.ztdb` file format is used for both Flexible Probe (dumping file declared by user) and Off-line Debugging (sniffer directory declared by user). When several instances of these features are used simultaneously, several files with this extension coexist and the ZeBu runtime library may create several hard links for them.

If these files are located on different disks/partitions (in particular if some are on a local partition and some are remote resources), the following error may be displayed:

```
ERROR : ZPRIV0062E : Cannot create link "[<second path>/]<second
name>.ztdb/<space file>" on "/[<first path>/]<first name>.ztdb/<space
file>", Invalid cross-device link
```

To avoid this error, it is recommended to declare paths on the same disk/partition for Flexible Probes dumping files and for sniffer directory.

5.7.7 Some initialization phases not carried out in specific C/C++ testbenches

When a C/C++ testbench controls the design using neither transactors nor co-simulation drivers (`HDL_COSIM`, `C_COSIM`, `MCKC_COSIM`), some of the initialization phases are not carried out, namely on memory. Under these circumstances, the design will not work even if no error is generated.

To avoid this, user must identify the testbench in the `designFeatures` file in the same way as is done for multi-process.

```
$nbProcess = 1;
$process_0 = "#process_name";
```

Note that the `#process_name` field must match the 3rd parameter of the open call (`zebu.work`, `designFeatures`, `process_name`).

This parameter not being mandatory, the default value can be used:

```
$nbProcess = 1;
$process_0 = "default_process";
```

At the beginning of the connection, an unidentified process will generate the following warning messages:

```
-- ZeBu : cpp_test_bench : WARNING : A process name has been specified "default_process" at
          open time, but it is not specified in the "./designFeatures" file.
-- ZeBu : cpp_test_bench : WARNING :     The list of specified process names is :
-- ZeBu : cpp_test_bench : Looking for a connection (pid 28241 at Tue 3 8 2010 - 17:46:57)
-- ZeBu : cpp_test_bench : "default_process" is a control-only process working on
          "../zebu.work".
```

An identified process will issue the following:

```
-- ZeBu : cpp_test_bench : Looking for a connection (pid 28200 at Tue 3 8 2010 - 17:46:11)
-- ZeBu : cpp_test_bench : "default_process" is a full-capability process working on
          "../zebu.work".
```

Unidentified processes can be used to investigate or control clocks, memories, logic analyzers, triggers, signals, etc. They cannot be used to control top-level IOs.

5.7.8 Clock declaration in the `designFeatures` file

5.7.8.1 Clock Declaration for HDL co-simulation

For HDL co-simulation, the clocks controlled by the HDL testbench must be declared in separate groups (one clock per group) in the `designFeatures` file. Additional design clocks which are not controlled from the HDL testbench can be grouped.

Note that this restriction also impacts the SVA feature, as described in Section 5.8.4.

5.7.8.2 Declaration of the waveform

Some unexpected initialization results on either registers or memories or mismatch with simulation may exist for some designs if the waveform of a controlled clock is declared for runtime with a first cycle at high level.

To inform user that a non-recommended waveform has been declared, the following warning message is displayed in the **`zServer`** log:

```
-- ZeBu : zServer : WARNING : The controlled clock "U0.M0.AAAA"'s waveform is "-_", it's not
recommended to start clocks with a high level.
```

Clock waveforms starting at a low level are strongly recommended to avoid the effects of the invisible rising edge when starting emulation.

5.7.9 Limitation on clock specification in the C/C++ testbench

When the clock parameters are declared in the C/C++ testbench **after initialization**, the testbench fails with the following error message:

```
ERROR : LUI1245E : Cannot treat "registerUserClock" request
ERROR : LUI1256E : You cannot register the clock "U0.M0.clk" when the clock system is already
enabled.
```

When restoring a saved state, it is impossible to declare the clock parameters in the C/C++ testbench:

- The declaration of clock parameters in the C/C++ fails with the same message if it is done **after initialization**.
- The testbench fails without any message when the clock parameters are declared in the C/C++ testbench **before initialization.**

As a workaround for restore, EVE recommends that the user declares clock parameters from the `designFeatures` file only (not from the C/C++ testbench).

5.7.10 Limitations for threadsafe environment

- The `Z_REPEAT_CALLBACK` is only available when emulating in a threadsafe environment (when testbench is linked to `libZebuThreadsafe.so` library).
- When running C/C++ co-simulation in a threadsafe environment with **`zRun`**, dumping a waveform file from dynamic probes with **`zRun`** becomes very slow.

5.7.11 Limitation on fast hardware state

A fast hardware state object cannot be initialized before board initialization.

5.8 Limitation for SystemVerilog assertions (SVAs)

5.8.1 SVA limitation when the design instantiates RLDRAM memory models

It is not possible to proceed with emulation runtime if SVAs were synthesized and the design instantiates RLDRAM memory models.

The following message is displayed at runtime because the ZeBu software enables SVAs by writing to registers, which is not supported when RLDRAM memory models are instantiated in the corresponding FPGA:

```
ERROR : ZHW0193E : Signal ZSVA.<name>.sva_enable is not writable.
```

5.8.2 Specific limitations for Full Report mode

The following limitations are applicable in Full Report mode (when the **Report only Failure** checkbox is not selected in **`zCui`**):

- Sequences:
  - Task/function calls with local variables as arguments are not supported.
  - Among sequence operators, multi-cycle sequences are supported for a maximum of 15 cycles.
  - Nested `first_match` and combination of `first_match` and implication is not supported.
- Properties:
  - Nested implication is not supported.
  - Combination of implication & `first_match` is not supported.
- Clock Detection:
  - Multi-clocked sequences combined with ##0 are not supported.
  - Multi-clocked properties with different leading clocks for sub-properties are not supported.
- Assert Control Tasks:
  - `$assertkill` & `$asserton` are not supported for multi-clocked assertions.
- Concurrent Assertions:
  - Assertions with a clocking event different from the clocking event of the procedural block are not supported

- Multi-clocked assertions are not supported in procedural block.
- Deferred Assertions:
  - Only supported in module & `always@(clocking event)` scope
- Immediate Assertions:
  - Only supported in module & `always@(clocking event)` scope

5.8.3 Specific limitations for Report only Failure mode

The following limitations are applicable in **Report only Failure** mode:

- Sequences:
  - Task/function calls with local variables as arguments are not supported.
  - Among sequence operators, multi-cycle sequences are supported only on the left side of an implication.
  - `AND` of unbounded assertion is not supported
  - `OR` of unbounded assertions supported only on left side of implication
- Properties:
  - `NOT` of property expressions not supported
- Concurrent Assertions:
  - `cover` supported only for bounded sequences.
  - Assertions with a clocking event different from the clocking event of the procedural block are not supported
- Deferred Assertions:
  - Only supported in module & `always@(clocking event)` scope
- Immediate Assertions:
  - Only supported in module & `always@(clocking event)` scope

5.8.4 Multiple clocks used by SVA vs. HDL co-simulation

Using the SVA feature with multiple clocks is supported (one single clock per assertion). However in order for SVA to work correctly with multiple clocks, the clocks used by SVA must be grouped in the `designFeatures` file.

A conflict arises when you use HDL co-simulation because multiple clocks must not be grouped in this case (the clocks are ungrouped by default). If the clocks are not grouped and you click the **SVA** button in the **`zRun`** GUI, the following error will be generated:

```
-- ZeBu : zRun : ERROR : ZRUN0118E : ZEBU_Sva_getSamplingClock : Clock group name not found
```

The same error will be generated if the `ZEBU_Sva_getSamplingClock` function is used in one of the following cases:

- Directly from the **`zRun` Tcl command** field:
  ```
  > ZEBU_Sva_getSamplingClock
  ```
- Via a Tcl script executed from the **Tcl command** field with the `-do` option:
  ```
  > zRun -do <my_script.tcl>
  ```

5.9 Restrictions related to RLDRAM memory models

Note that the limitations listed in this section are not applicable for ZeBu-Server systems with only 16C FPGA modules.

5.9.1 Restrictions on writing to registers

For most FPGAs in the system, writing to registers at runtime is supported when the **Enable BRAM Read&Write / Write Register / Save&Restore** item in the **Debugging** tab is enabled in **`zCui`** (it is enabled by default).
However, it is never possible to write to registers in FPGAs connected directly on RLDRAM in 4C and 8C/ICE modules, when RLDRAM is used by the design.

This restriction impacts the following commands of the RTL-based Compilation Scripts when they apply to signals/instances mapped into an FPGA connected to RLDRAM (no message is displayed during compilation; the write access limitation only appears during emulation runtime):

- `force_dyn`
- `force –value REG`
- `blackbox –value REG`

This restriction also impacts the SVA feature, as described in Section 5.8.1.

5.9.2 Restrictions on reading/writing to BRAM memories

Writing to BRAM memories at runtime is supported with the following restrictions:

- If neither the **Enable BRAM Read&Write / Write Register / Save&Restore** nor the **BRAM Instrumentation for Read&Write** items in the **Debugging** tab have been set, then at runtime, all output registers of all BRAMs in a FPGA will be reset each time one BRAM is read or written on that FPGA, causing corruption of the data read from the memory.

- Setting the **Enable BRAM Read&Write / Write Register / Save&Restore** item can avoid this behavior, except for FPGAs connected directly on RLDRAM in 4C or 8C/ICE modules, when RLDRAM is used by the design. To avoid the behavior described above for these FPGAs, select **BRAM Instrumentation for Read&Write**.

5.10 ZEMI-3 limitations

- The XST synthesizer is not supported for ZEMI-3 transactors.
- It is not possible to synthesize a ZEMI-3 transactor in top-down mode (block-based synthesis is mandatory).
- A ZEMI-3 transactor cannot be called with a name from the ZEMI-3 public API, such as `reset`, `terminate` or `disconnect`. This API is described in the `$ZEBU_ROOT/include/ZEMI3Handler.hh` header file.

5.11 Advanced triggers

Advanced triggers are not supported in the current version.

5.12 ZW-FPGA with **zFAST** synthesizer

In the present software version, the following models are not supported by ZW-FPGA when synthesizing with **zFAST**:

- `DW_asymfifoctl_s1_df` (DW03)
- `DW_asymfifoctl_s2_sf` (DW03)
- `DW_arbiter_2t` (DW05)
- `DW_arbiter_dp` (DW05)
- `DW_arbiter_fcfs` (DW05)
- `DW_asymfifo_s1_sf` (DW06)
- `DW_asymfifo_s2_sf` (DW06)

The complete list of the supported models, not taking into account the present limitation, is available in the dedicated Application Note (AN022, Rev. D) in the present documentation package.

5.13 Deprecated tools for simulation-like command line interface

The **zRFEvlog**, **zRFEvhdl** and **zRFEelab** tools are no longer delivered with the V6_3_1 software package. These tools provided a simulation-like command line interface to declared RTL source files for a third-party synthesizer with **zRtlFrontEnd** or for **zFAST**.

For a similar feature, the user should select the **zFAST** script mode when synthesizing with **zFAST**. This mode is described in the *zFAST Synthesizer Manual*, Rev. A.

When synthesizing with a third-party synthesis with **zRtlFrontEnd**, the script with simulation commands has to be declared in **zCui** in the RTL group as a source file with **Add Command File**s item.

6 Fixed Issues

The following issues are now fixed, either in an intermediate patch for V6_3_0 or more recently for the present V6_3_1 software release.

In order to improve legibility of this Release Note, some minor corrections introduced in the present software release are not listed.

6.1 Compilation interface (`zCui`)

This release fixes the following **`zCui`** issues:

- Compiling with **`zCui`** graphical interface was not possible on a PC which did not use Red Hat Enterprise Linux 4 operating system. The compatibility issue for the graphical libraries has been fixed and **`zCui`** now works properly also on Red Hat Enterprise Linux 5 and SUSE 10 operating systems.
- When a compilation is stopped from the **`zCui`** interface, a 120 seconds timeout now exists so that all the jobs are killed, at the latest after the timeout:
  - If no job is running when the timeout is reached, **`zCui`** can be used right away.
  - If one or more jobs are still running, a warning message is displayed so that user can check why the jobs did not terminate as expected. This message must be acknowledged by user.
- The text selection feature in a log displayed in **`zCui`** was limited to entire lines. It now supports line portions, including multi-line selections starting anywhere in the first line and finishing anywhere in the last line.
- In some cases the `pending` status icon remained for some tasks after the compilation had been stopped by the user; reported in RT#25598. Now, only the tasks actually processed have a status icon (either `passed` or `failed`).
- The font size for the logs displayed in **`zCui`** is now saved when closing **`zCui`** so that it remains the same next time **`zCui`** is launched; reported in RT#25619.
- For a ZEMI-3 transactor, **`zCui`** did not take into account the `-v`, `-y` and `libext` options for the Verilog libraries; reported in RT#25677.
- The title of the log pane for FPGA compilation tasks (**Controller**, **ISE**, or **Finish** tasks) did not display the name of the FPGA but only the short name of the task. It was not easy to know for which FPGA was a given log, in particular when the task tree was collapsed; reported in RT#26302. The complete name of the task, including the name of the FPGA, is now displayed.
- When an FPGA P&R task (ISE tasks) was selected, the log which was displayed (with both **Show log in place** and **Show log in window**) was not the `compilation.log` of ISE as expected; reported in RT#26398.

- When an **Output Mapping File** was declared by the user in the **Clustering** panel of **zCui**, this file was inappropriately used as a zCore declaration file by the system-level compiler; reported in RT#26464. This dependency has been removed and this file is now used only for clustering purposes.

6.2 zFAST synthesis

This release fixes the following **zFAST** issue:

- When synthesizing in top-down mode, the registers of the design can now be accessed in the runtime database.
- Incremental compilation has been improved when synthesizing with **zFAST**, and is now the same for both Script and Standard modes:
  - If a used file in a library directory or in an include directory changes, the design is re-analyzed and re-elaborated.
  - If a new file is added in a library directory or in an include directory, the design is not re-analyzed and re-elaborated.
- When the number of ports per memory instance was too high, the synthesis failed because of the number of signals; reported in RT#25889. The following message was displayed:

```
###fatal error in ZFAST [110.5372] : No. of terminals exceeds the maximum limit of
65365 that can be handled
```

  The limit has been increased to 256K terminals (64K ports with 4 signals) per instance which is high enough to avoid similar issues.
- When one-hot decoders were implemented by shifting constant 1, the synthesis was not correct in case of operand width mismatch; reported in RT#26435. For example shifting by a 3-bit value and assigning it to a variable that was wider than 8 bits. This caused some mismatches during runtime.
- When the bit streaming operator (<<) was used in SystemVerilog source files, synthesis failed with the following messages; reported in RT#26279:

```
INFO: zFe: execing Tcl status is 1
INFO: zFe: execing failed: error type CHILDSTATUS: child process exited with non-zero
      status = 1.
INFO: zFe: child process had PID = 6674
INFO: zFe: execing failed: command did not correctly finish: child process exited
      abnormally
```

- When a macro in the customer source files expanded to the following code, it was not synthesizable. Reported in RT#25931 and RT#26137:

```
always @(posedge clk or posedge reset)
  if (1'b0) q = 1'b1;
   else if (reset) q = 1'b0;
     else if (en) q = data;
```

  By setting the `Compile:FixIfConst=true` attribute for **zFAST**, the RTL source is now transformed into synthesizable code:

```
always @(posedge clk or posedge reset)
  if (reset) q = 1'b0;
   else if (en) q = data;
```

  This transformation may cause errors in case of hierarchical references.

- When an RTL element synthesized as a **zMem** memory was retrieved in the C++ testbench by its RTL name using the `getSignalFromPath` method, the `ZEBU::Signal::Force` method could not be used and the runtime failed with the following message; reported in RT#26368:

```
ZeBu : WARNING : Cannot find scope node (.top)
ZeBu : ERROR : ZBSE0149E : The signal '.top.array[0][0][1:0]' cannot be found.
```

  The problem is now fixed and the force works as expected at runtime.

- When a hierarchical reference in the design referenced a variable in the target module, which was also of a type `local` to the target module, then in some cases the href processing left the design in an inconsistent state; reported in RT#26082. This caused synthesis to report a syntax error for undeclared variables:

```
### fatal error in ZFAST [97.606] : <source.sv:460>:  Identifier ( _tmp_XXX ) not
    declared in current scope.
```

- During VHDL parsing the following error was thrown incorrectly, as reported in RT#26880:

```
### fatal error in ZFAST [115.312] : <file:line>: Parameter kind of AAA does not
                                     match with declaration
### fatal error in ZFAST [115.271] : <file:line>: Subprogram body header and
                                    corresponding declaration do not conform lexically
```

- When synthesizing the following source code:

```
module test(
input wire[32'hFE:0] in_data,
output reg [32'hFE:0] out_data) ;

localparam START_BIT = 32'd10 ;
localparam END_BIT = 32'd20 ;
localparam STEP = -1 ;

always @(*) begin : out_proc
 integer ii ;
 reg [0:0] count_down ;
 reg one_seen ;
 count_down [0] = 1'b0 ;
 one_seen = 1'b0 ;

 for (ii=START_BIT ; $unsigned(ii) < END_BIT ; ii = (ii + 1)) begin
  out_data[ii] = ((~one_seen) & (~count_down[0]));
 end
end
endmodule
```

  the following error message was displayed, as reported in RT#27082:

```
### fatal error in ZFAST [110.5233]: <../../../design.sv:17>: For loop not unrollable
```

- Using `$typeof` with unpacked arrays was generating the following unclear error, as reported in RT#27231:

```
### fatal error in ZFAST [97.1932] : <TeCreate.cc:1779>: Error occurred to create
object (VE_ARRAY) by the PI (veCreateNormalArray).
```

  Now when using `$typeof` with unpacked arrays the return value of the function is 0 and the following warning is displayed:

```
### warning in ZFAST [39.268] : <../../../design.sv:5>: Task (Function) '$typeof' not
                                supported; will be ignored (replaced with 0)
```

- The following code caused a segmentation fault with stack trace during synthesis because of the use of an integer as `loop_idx[31:0]`, as reported in RT#27103:

```
integer loop_idx ;
always @(*) begin
 for (loop_idx = 0 ; (loop_idx[31:0] < DEPTH ) ; loop_idx =
(loop_idx+1))
 reqbuff_nxt[loop_idx] = reqbuff_r[loop_idx];
end
```

6.3 Back-End Compilation

6.3.1 Declarations with RTL paths

In the RTL-based compilation script, the `force` command now supports `-rtlname_input/-rtlname_output` options.

6.3.2 Generation of co-simulation wrappers

In the previous release, it was not possible to generate HDL co-simulation wrappers with the `-bb` option in **`zDbPostProc`**/**`zSelectProbes`** (presence of a blackbox).

6.3.3 System-level compilation

This release fixes the following system-level compilation issues:

- When a zCore was connected to the Direct ICE interface and had to be spread among several modules, the automatic FPGA allocation in the system-level compiler entered in an infinite loop, as reported in RT#27073.

- When the `use_fpga` command was declared in the script for the system-level compiler for an nonexistent zCore, the system-level compiler created an inappropriate script file for the zCore-level compiler as if the zCore was present; reported in RT#26177. Because there was no netlist corresponding to this zCore, **`zCui`** reported a `Missing Input` error for this zCore-level compilation task. With this release, the system-level compiler ignores the `use_fpga` commands in such conditions.

- When optimizing the interface between zCores, the system-level compiler inappropriately removed some elements for flexible probes insertion; reported in RT#26307. The zCore-level compiler thus failed to insert the flexible probes, and displayed the following warning message:

```
### warning in FLEXIBLE_PROBES [TRC0089W] : Wire 'top.OUT2' does not have a driver
    while probing object 'top.OUT2' in command 'probe_signals -port {top.OUT2}
    -hier_sep . -type flexible -fifo low -group grp_xyz'. Discarding it.
```

  The system-level compiler has been fixed so that the flexible probes are now inserted correctly.

- When optimizing the interface between zCores, the system-level compiler inappropriately merged some signals and failed with the following error message; reported in RT#25986:

```
### internal error [PRO0601E] : Pre (3) and post (23) driver replication reader count
                                differs for wire 'a.b.c[1]'
### internal error [PRO0601E] : libNP_SplitNetlist.cc, line 3403
```

- Due to an issue in the system-level compilation flow, dynamic forces caused some signals to be displayed without values (`x`) in the **Dump** and **Monitor** windows; reported in RT#26491. In order to take all user forces and probes into account, the runtime database is now only generated after they have all been applied.
- While optimizing split results, the algorithm which moves accessibility resources from one zCore to another crashed with the following internal error and stack trace; reported in RT#26657:

```
### Internal error (ref '11')
  # step STACK : [ 3 ] : 0x107002b – zTopBuild
    (_ZNK10znl_Module17GetInstanceNumberEm+0xb) [0x107002b]
```

- When dynamic forces were applied on latches, the output signals of these latches were no longer writeable; reported in RT#26723. Latches are now correctly processed, making their output signals both forceable (controlled manually all the time until released) and writeable (written at one cycle and then working as in the design) at runtime. Also, more information is displayed on dynamic force drivers.
- When removal of loadless logic was activated in system-level compilation (via the `rm_loadless_logic` command):
  - SVA blackboxes generated by **`zFAST`** were deleted, thus causing the inoperability of SVA feature at runtime.
  - `iobufs` were not correctly handled, resulting in System Place and Route errors generated for undriven wires.
- When a tristate bus was driven through `assign` statements in the RTL code but some of these assignments were not actually driven, the system-level compiler failed with the following internal error; reported in RT#26719:

```
### internal error [ZTB0192E] : Bus 'sys.s0.nec.giolgc.giolgcbclka .xxBclkInN' still
    has synthesis buffers as driver: sys.s0.nec.giolgc.giolgcbclka.sqnod19.O.
### internal error [ZTB0192E] : ztb_assignDB.cc, line 1162 : checkBuses
```

6.3.4 zCore-level compilation

This release fixes the following zCore-level compilation issues:

- Because the automatic clustering in the zCore-level compiler sometimes did not process correctly the specific flexible probes for DPI, the compilation of FPGA failed with the following error message; reported in RT#26309:

```
### fatal error in FLEXIBLE_PROBES [TRC0003E] : Limit of 64 physical tracers has been
    reached (312 physical tracers are used).
### tcl error in zNetgen [CCT0104F]
```

  Note that the following information was displayed in the `Build zCore` log file but without causing an actual error in this task:

```
# step AC : FLEXIBLE PROBE allocation (1 nodes with probes/18 nodes, 1 overflow(s)):
(...)
# step AC : **********************************************************
# step AC : * CLUSTERING RESULT SUMMARY (cf. above for detailed allocation) *
# step AC : *                                                        *
# step AC : * LOGIC MAPPING - OK                                     *
# step AC : * ZRM MAPPING - OK                                       *
# step AC : * FLEX. PROBES MAPPING - ERROR                           *
# step AC : **********************************************************
```

- The zCore-level compiler inappropriately removed zrm memory instances from the netlists when the outputs of these memories were not connected to outputs of the design or to dynamic probes; reported in RT#26141. Such memory instances are now kept in the netlists after zCore-level compilation.
- When **Filter Glitches** was selected, some signals were sometimes delayed in the waveform files at runtime because of cascaded asynchronous set/reset in the design, as reported in RT#27166.

6.3.5 Performance Oriented Partitioning

This release fixes the following issue for performance-oriented partitioning:

- When **Performance Oriented Partitioning (zCore-level compilation)** was selected in **`zCui`**, the cut estimation was not correct for system cells (specific cells added by the ZeBu compiler) and the resulting partitioning solutions were not optimized when a high number of such cells were present. A similar issue also existed for nets connected to empty instances.

6.3.6 Density Oriented Clustering

This release fixes the following issues for density-oriented clustering:

- zCore-level compilation failed with the following internal error when some cores were defined with `cluster core add` and some blocks including zrm were merged with `cluster merge_blocks`; reported in RT#26688:

```
### internal error [CLU0541E] : group_definition::get_group_definition: cannot find
                                group post_vec_merged_group_xxx
```

- When the `cluster core` command was declared for the zCore-level compiler, some FPGA resources did not appear correctly in the table summarizing sizes, constraints and resource usage.
- Automatic clustering failed with the following fatal error when some mapping commands were declared for zrm memories in the zCore-level script; reported in RT#26555:

```
### fatal error in AC [AC00092F] : Cannot find enough resource for memories
```

6.3.7 System Place and Route

This release fixes the following System Place and Route issues:

- Some nets declared as fast nets were not accelerated due to a bad construction of communication sockets; reported in RT#26556. That sometimes resulted in a difference between the frequency computed by **`zTime`** and the real runtime frequency.
- Bad renaming of nets which were to be routed by System Place and Route caused the following internal error. Reported in RT#25942, RT#26717, and RT#26831:

```
fatal error in CHECK [KPR0436F] : In wrapper(0,9), no source for wire
               AA@B.CC.DD.EE.FF.GG.HH.II.JJ[12]__zpar_rename
fatal error in CHECK [KPR0436F] : In wrapper(0,9), no source for wire
               AA@BB.CC.DD.EE.FF.GG.HH.II.JJ[12]__zpar_renameAA@BB.CC.DD.EE.FF.GG.HH.
               II.JJ[12]__zpar_rename0
error in UNCREF : Found 2 errors, cannot continue execution.
```

6.3.8 FPGA Compilation

This release fixes the following issues for FPGA compilation:

- When the PARFF feature was activated, compiling on a PC farm sometimes caused some unexpected failures during FPGA Place & Route because some supervision processes were killed with no reason; reported in RT#26477.
- In some cases, mapping **`zMem`** memories in IF FPGAs caused the following error message during FPGA compilation (`ngdbuild` step). Reported in RT#26734:

```
ERROR:NgdBuild:604 - logical block 'abcdef' with type 'zxlsys' could not be resolved.
      A pin name misspelling can cause this, a missing edif or ngc file, or the
      misspelling of a type name. Symbol 'zxlsys' not supported in target 'virtex5'.
```

- When the PARFF feature was activated, the parameters were loaded from the parameter file only once when launching **`zCui`**, as reported in RT#27437. Any modifications in the file or the declaration of a **Superseded ParameterSet File** were therefore not taken into account and **`zCui`** had to be relaunched. With V6_3_1, the PARFF parameters are updated for each compilation.
- When the PARFF feature was activated, Xilinx ISE 'par' sometimes lasted more than 1h but the supervision task did not decide to relaunch a new attempt, as reported in RT#27774. This relaunch is now performed when the duration of 'par' is more than 1h.

6.3.9 Static Timing analysis

In some cases, the HTML reports for Static Timing analysis (**`zTime`**) reported incomplete names for the paths in the set/reset asynchronous paths analysis. However, the estimation of the runtime parameters for clock modeling was correct.

6.3.10 Database Generation

Vectorization was not properly done when creating the runtime database for an RTB, causing the following error message at runtime; reported in RT#26471:

```
ZeBu : zRun : ERROR : ZPRIV1610E : Vector Vavcw3(4] getLeft is NULL.
```

6.4 Combinational simulated signals (CSA)

This release fixes the following issues for Combinational Simulated Signals (CSA):

- When some vectors had some bits declared in `force_dyn` commands and others computed by CSA, it was not possible to force the signals at runtime because the vectors were incorrectly considered as fully computed by CSA; reported in RT#26723. The following message was displayed:

```
ERROR: Failed to aquire runtime Signal for signal_path[value]
ERROR: Part create failed (1,32,0)
```

  Such vectors are now considered as non-CSA to support the force on the bits which are not computed by CSA.

- When processing the CSA signals, some vectorization issues in the database caused the following error message; reported in RT#27010:

```
### fatal error in SCS [SEG0143F] : Cannot find wire <wire_name> in the graph
    <graph_name>
```

- A segmentation fault sometimes occurred during the Restore when some CSA signals were selected.

6.5 Runtime

This release fixes the following runtime issues:

- When launching **`zInstall`**, the user name (`root`) could not be retrieved, as reported in RT#27331. The following error message was displayed:

```
zInstall: creating link /zebu/queue/queue..259.259.0.130551 -> /proc/259
zInstall: ERROR: LUI0948E : Cannot extract information from "queue..259.259.0.130551"
```

- When the loading from the bitstream cache was stopped with a Ctrl+C, the stored bitstream was corrupted and the next load from the cache failed with the following error message:

```
Error : LKAU0159E : FPGA xxxx  is not correctly downloaded
```

  The design was then loaded from files which takes more time. Such corruption of the cache no longer exists and the message when switching to **From Files** mode has been changed:

```
Warning : Download Ux_Mx_Fxx from cache failed : retrying from file ...
```

- The inter-FPGA communication clock (`xclock`) frequency resulting from compilation was not checked before selecting the load from the cache or from files. It could result in loading inappropriate bitstream files for the selected `xclock` frequency.

- Because `RAMB36SDP` and `RAMB18SDP` were not correctly processed for delay insertion, some races between data and clock signals were observed at runtime and, in some cases, made read/write accesses impossible or caused corrupted data when read/written in the memory. Reported in RT#25819, RT#26041, and RT#26221.

- Some messages from the hardware were not stored correctly in the Save operation and caused failures when restoring the design. Note that after this patch, it is no longer possible to restore a database created with a former software version.

- Parallel loading of FPGAs sometimes failed while initializing ZeBu-Server; reported in RT#25270:

```
ZeBu: zServer: Loading U0_M0_F00 with "XXX/design.bit" (using CPU).
ZeBu: zServer: WARNING : Cannot read on channel 1 (software timeOut)
```

  The parallel mode is no longer the default and the FPGAs are now loaded in a serial mode. However with the use of the bitstream cache, the serial mode does not increase the loading duration.

- When Ctrl+C was pressed while loading the FPGAs (for diagnostics or runtime purposes) the PC hung and stopped the FPGA loading; reported in RT#24514.
- When loading the FGPAs for diagnostics or for emulation runtime, the messages displayed on the standard output sometimes stopped but this did not actually impact the FPGA loading; reported in RT#25688.
- When the `libZebuRestore` library was used, some signals were not explicitly set as writeable in all their occurrences in the database and they were sometimes erroneously reported as non-writeable. In such cases, the signals could not be written at runtime, causing issues when restoring or comparing a previously saved state.
- When synthesized with **`zFAST`**, multi-dimensional arrays were not correctly processed and sometimes resulted into erroneous signals in the runtime database which caused non-deterministic behaviors with CSA feature activated; reported in RT#27020. This correction requires relaunching the synthesis of the design.
- A stuck signal or a vector with all its bits stuck were erroneously reported as writeable in the runtime database; reported in RT#27021. Stuck signals are now correctly reported as non-writeable. Vectors are writeable when they have at least 1 writeable bit and other bits are either stuck or writeable. If all the bits of a vector are stuck or if 1 or more bits are read-only, the vector is read-only.
- Because they may impact the runtime performance and to improve the legibility of the vectors, very large vectors (width > 4096 bits) are now displayed as separate bits.
- In C/C++ co-simulation, the timestamps of a waveform file always started from cycle 0, without taking into account the number of cycles already run before dumping; reported in RT#25191.
- When dumping an FSDB waveform file, the data of the first cycle were not available because they were overwritten by the next cycle (the waveforms were shifted by 1); reported in RT#26941.
- When a ZTDB waveform file was corrupted or incomplete, the **`ztdb2vcd`**, **`ztdb2fsdb`** and **`ztdb2prd`** conversion tools sometimes ran forever; reported in RT#26472. They now display the following error message when that happens:

```
FATAL : ZREST0752E : Cannot read .ztdb file, "flp.ztdb/main", abnormal current
        cycle=669293383 <> expected cycle=675747036
ERROR : ZREST0752E : Cannot read .ztdb file, "flp.ztdb/main", abnormal current
        cycle=669293383 <> expected cycle=675747036
```

- When the `setMsgVerboseMode()` method was used to hide messages, some of the messages were not hidden as expected, in particular error messages on triggers.
- When investigating runtime issues with Valgrind, the following errors were sometimes reported for the ZeBu software without impacting the emulation runtime; reported in RT#26579:

```
# step DB LOAD : ZebuDB loaded in 8 seconds
==13800== Invalid read of size 1
```

- When the selected signals were dumped with `dumpvars(NULL)` some memory consumption issues occurred; reported in RT#26456.
- When a signal and a vector were homonyms in an instance of the database, a warning is now displayed to make investigation easier since it may cause runtime issues:

```
Cannot add the signal/vector '<sig_name>' under the node '<node path>'.
A signal/vector with the same name has already been added.
```

or

```
A signal and a vector share the same name '<sig_name>' in the instance '<inst_path>'.
Priority is given to the signal.
```

- The delay before returning an error when **`zServer`** connects to the ZeBu system has been increased to 120 seconds in order to avoid the following error message; reported in RT#26692:

```
testbench : ERROR : LUI0972E : Cannot detect server which registered design (#pid).
```

- In a multi-thread environment, accessing flexible probes sometimes caused segmentation faults because of concurrent accesses; reported in RT#26518.
- When **Offline Debug** was activated, some issues were faced at runtime because some inout or tristate ports/nets declared at the interface of the DUT were also declared in the DVE file for use as static probes; reported in RT#26245. Those ports/nets were not correctly connected in the RTB.
- When the sampling clock for flexible probes was declared in the DVE file, the **Flexible Local Probes** panel in **`zRun`** did not open and a Tcl error message was displayed in a pop-up.
- When **`zRun`** was used with a testbench, saving the state of the design sometimes caused a timeout error on read access either in **`zServer`** or in the testbench; reported in RT#25808 (note that this error was inappropriately reported as a warning but the emulation was actually stopped):

```
ZeBu : WARNING : Cannot read on channel x (software timeOut) at <info about address>
```

6.6 System Verilog Assertions

This release fixes the following issues for System Verilog Assertions:

- In the ZeBu runtime libraries, the `ZEBU_SVA_SelectReport` function of the C API was not available (only the corresponding C++ method was available).
  The interface for this C function is:
  ```
  int ZEBU_SVA_SelectReport(Board *, const ZEBU_SVA_Severity)
  ```
- Because of incorrect processing in the system-level compiler when SVAs were selected, the activation of the Trigger Mechanism for SVAs at runtime did not work properly.
- The system-level compiler did not initialize some values for the zCore-level compiler, leading to the tracing of some bits which were not meant to be traced.
- Because the files for SVA selection were sometimes generated empty by the system-level compiler, the zCore-level compiler failed with the following error messages:
  ```
  ### fatal error in SVA  [TRC0080W] : No instance is matching exclude patterns.
  ### fatal error in SVA  [TRC0081W] : No instance is matching select patterns.
  ```
  Note that these files were also stored in an inappropriate directory, causing inconsistencies when multiple back-ends were used.
- The runtime no longer fails when the `SVA::Start` method is used in the testbench but the design has no SVA or when SVAs were not synthesized.
- To avoid manual removal of unsupported assertions in the RTL source files because they caused some synthesis errors, they can now be ignored by the synthesizer with the following attribute:
  ```
  SVA:KillUnsupported = true
  ```
  A warning is displayed during elaboration.
- When the **System Verilog Assertions** checkbox was selected in **`zCui`** when synthesizing the design, the activation of SVAs at runtime caused a fatal error when no SVAs were actually present in the runtime database.
  Starting with V6_3_1, no error is reported for the same situation.

6.7 ZEMI-3 transactors

This release fixes the following issues with ZEMI-3 transactors:

- If a transactor contained function calls in a structural context, e.g. in a continuous `assign`, then in some cases **`zemicomp`** failed with the following error message; reported in RT#25675:

```
vhs: intval.cc:116: TeNode* ZemiBehav::IntValWrap::getPrev(const TeNode*):
                    Assertion 'par' failed.
```

- If a variable was declared to be of a type defined in a package and imported using import `pkg::*`, then **`zemicomp`** created some erroneous source, causing the following syntax error in **`zFAST`**; reported in RT#25762:

```
### fatal error in ZFAST [97.606] : <../zxtor/xtor/zxtor.xtor.v:1234>:
                                    Identifier ( XXX ) not declared in current scope.
```

- If a transactor contained a `case` statement with no default clause, then in some cases **`zemicomp`** failed with the following error message; reported in RT#25762:

```
vhs: ZemiBehav/controlflow.h:382: ZemiBehav::CFG<Lattice>::~CFG()
     [with LatticeT = ZemiBehav::DFEvent]: Assertion '_dfStack.size() == 0' failed.
```

- When the inputs of an exported DPI function were unused, or if they drove dead logic, then the message signal can get optimized away during synthesis, leaving an unconnected port for the `zceiMessagePort` macro; reported in RT#26298. This caused the following errors in back-end tools (in system-level or zCore-level compiler):

```
### fatal error in pcubfm_if0 [DRV0163F] : input message port is not connected for
                                           instance 'XXX'
```

  `zemicomp` now processes such signals correctly so that they are not simplified during synthesis.

6.8 ZEMI-3 and zDPI

- When using multiple context import functions in the same testbench and performing concurrent calls, the `svGetNameFromScope` function sometimes returned a corrupted `const char *` value (the returned value was for a different DPI function call).

- When synthesizing with **zFAST Script Mode** and the zDPI feature, the specific files generated by **`zFAST`** (`grp0_ccall.*`) were not taken into account by the system-level compiler and were not copied in the `zcui.work/zebu.work` directory as expected for the compilation of the DPI functions; reported in RT#25954.

7 Version Compatibility

This chapter includes information which is applicable for Version 6_3_0.

7.1 Supported PC configurations

The ZeBu compilation and runtime tools are 64-bit software which can be used only on a 64-bit PC configuration (this release does not support 32-bit PCs).

This version of ZeBu has been successfully tested on the following PC configurations:

- Compilation:

Dell PowerEdge 1950	(RHEL 4.5)
Dell PowerEdge 2950	(RHEL 4.5)
Dell PowerEdge R410	(RHEL 4.5)

- Emulation Runtime:

Dell Optiplex 760	(RHEL 4.7)
Dell Optiplex 760	(RHEL 4.8)
Dell Optiplex 760	(SUSE 10)
Dell Precision T5500	(RHEL 5.4)
HP xw6200	(RHEL 4.6)
HP z400	(RHEL 5.4)
HP z800	(RHEL 5.4)
HP DL580G7	(RHEL 5.5)
HP xw4550	(RHEL 5.1)

7.2 Operating system compatibility

EVE recommends using the following operating systems (installed in 64 bits) which has been successfully tested for both compilation and emulation runtime:

- Red Hat Enterprise Linux 4 operating system, kernel 2.6.
- Red Hat Enterprise Linux 5, kernel 2.6. See specific limitations in Section 5.1.2.
- SUSE Linux Enterprise Server 10, kernel 2.6. See specific limitations in Section 5.1.1.

7.3 Hardware compatibility

- This release is compatible with ZeBu-Server only.
- It is not compatible with ZeBu-XL, ZeBu-XXL, ZeBu-UF, and ZeBu-Personal.

7.4 Software compatibility

Detailed information about software compatibility and installation recommendations is available in Chapter 1.

7.5 ZeBu licenses

License features are the same as for ZeBu-Server version 6_3_0. Licenses which were used on ZeBu-XXL/UF/Personal cannot be used on ZeBu-Server.

No new licenses are required to support the present software release.

Specific licenses are required for:

- Xilinx ISE 11 software (mandatory).
- Concept Engineering netlist analyzers (optional, EVE can provide evaluation licenses for this product).

Note that, starting with the present software release, the ZeBu license daemon is also available for Red Hat Enterprise Linux 5.

You should contact your usual EVE representative for detailed information.

7.6 Third-party tools compatibility

The following table gives information about the third-party tools that have been successfully tested by EVE with the present version of the ZeBu software. This list is given for information purposes and does not necessarily imply that other versions will cause system malfunction.

Tool	Tested Versions	Tested OS compatibility *	Remarks
Synplify Pro	2010.09-SP1	RHEL 4.5/5.4 SUSE 10	FPGA synthesizer. http://www.synopsys.com
XST	11.4i	RHEL 4.5/5.4	FPGA Synthesizer. http://www.xilinx.com
VCS	mx-2009.06	RHEL 4.6/4.7/4.8/ 5.4 SUSE 10	HDL Simulator. http://www.synopsys.com
ModelSim	6.6b	RHEL 4.6/4.7/4.8/ 5.4 SUSE 10	HDL Simulator. http://www.mentor.com
NC-Sim	Not Tested		HDL Simulator. NC-Sim should work since ZeBu uses the standard PLI. http://www.cadence.com
SystemC	2.2.0	RHEL 4.6/4.7/4.8/ 5.4 SUSE 10	Required for SystemC co-simulation. http://www.systemc.org
GTKWave	3.1.11	RHEL 4.5	Waveform viewer. http://gtkwave.sourceforge.net Included in ZeBu software package.
Debussy	2009.10	RHEL 4.6/4.7/4.8/ 5.4 SUSE 10	Interface for design debug and waveform viewing. http://www.springsoft.com/
gcc	3.4.6	RHEL 4	C compiler. Part of the Linux operating system distribution. Required for C/C++ and SystemC co-simulation. May be required during installation process in case of kernel compilation. http://gcc.gnu.org/
	4.1	RHEL 5 SUSE 10	For runtime on a RHEL 5 operated PC, the testbench and the software part of the transactor should be compiled with this version of gcc C/C++ compiler.
ISE Place & Route	11.4i_patched	RHEL 4.5	Xilinx tools for FPGA Place & Route during ZeBu compilation. http://www.xilinx.com Included in ZeBu software package.
RTLvision PRO GateVision PRO	5.0.0	RHEL 4.5	RTL and gate-level netlist analyzers. http://www.concept.de Included in ZeBu software package.

OS compatibility may change. Please visit third-party websites for information.

7.7 Diagnostics patches

Diagnostics patches are specific to each release and to each type of system (number of units and number/types of FPGA modules). **`zUtils`** uses them to check your system once installed. To reduce the size of the ZeBu software package, they are not included and should be downloaded as shown in *ZeBu-Server Installation Manual*.

7.7.1 Diagnostics patches for configurations with up to 5 units

See file naming format for diagnostics patches in *ZeBu-Server Installation Manual*.

7.7.1.1 2-slot unit in single-unit configuration

```
zebu_V6_3_1.patch_0.diags_202_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_202_43_43.V50.sh.gz
zebu_V6_3_1.patch_0.diags_202_91_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_202_91_43.V50.sh.gz
zebu_V6_3_1.patch_0.diags_202_91_91.V50.sh.gz
zebu_V6_3_1.patch_0.diags_202_93_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_202_93_43.V50.sh.gz
zebu_V6_3_1.patch_0.diags_202_93_93.V50.sh.gz
```

7.7.1.2 5-slot unit in single-unit configuration

```
zebu_V6_3_1.patch_0.diags_501_161_161_161_02_02.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_161_161_161_161_02.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_161_161_161_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_43_161_161_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_43_43_43_02_02.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_43_43_43_43_02.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_43_43_43_43_43.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_91_161_161_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_91_43_43_43_43.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_91_91_91_02_02.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_91_91_91_91_02.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_91_91_91_91_91.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_93_161_02_02_02.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_93_161_161_02_02.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_93_161_161_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_93_43_43_43_43.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_93_93_161_02_02.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_93_93_161_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_93_93_93_02_02.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_93_93_93_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_93_93_93_93_02.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_93_93_93_93_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_501_93_93_93_93_93.V50.sh.gz
```

7.7.1.3 Single 5-slot unit in multi-unit configuration:

```
zebu_V6_3_1.patch_0.diags_C00-00-512_161_161_161_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C00-00-512_43_43_43_43_43.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C00-00-512_91_161_161_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C00-00-512_91_91_91_91_91.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C00-00-512_93_161_161_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C00-00-512_93_93_93_93_93.V50.sh.gz
```

7.7.1.4 Multiple 5-slot units in multi-unit configuration (up to 5 units):

```
zebu_V6_3_1.patch_0.diags_C00-02-512_161_161_161_161_161-
                                 512_161_161_161_161_161-
                                 512_161_161_161_161_161-
                                 512_161_161_161_161_161-
                                 512_161_161_161_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C00-02-512_161_161_161_161_161-
                                 512_161_161_161_161_161-
                                 512_161_161_161_161_161-
                                 512_161_161_161_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C00-02-512_161_161_161_161_161-
                                 512_161_161_161_161_161-
                                 512_161_161_161_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C00-02-512_161_161_161_161_161-
                                 512_161_161_161_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C00-02-512_161_161_161_161_161-
                                 512_91_161_161_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C00-02-512_91_161_161_161_161-
                                 512_161_161_161_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C00-02-512_93_161_161_161_161-
                                 512_93_161_161_161_161-
                                 512_93_161_161_161_161-
                                 512_93_161_161_161_161-
                                 512_93_161_161_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C00-02-512_93_93_93_93_93-
                                 512_93_93_93_93_93-
                                 512_93_93_93_93_93-
                                 512_93_93_93_93_93-
                                 512_93_93_93_93_93.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C00-02-512_93_93_93_93_93-
                                 512_93_93_93_93_93-
                                 512_93_93_93_93_93-
                                 512_93_93_93_93_93.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C00-02-512_93_93_93_93_93-
                                 512_93_93_93_93_93-
                                 512_93_93_93_93_93.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C00-02-512_93_93_93_93_93-
                                 512_93_93_93_93_93.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C02-02-512_161_161_161_161_161-
                                 512_161_161_161_161_161-
                                 512_161_161_161_161_161-
                                 512_161_161_161_161_161-
                                 512_161_161_161_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C02-02-512_161_161_161_161_161-
                                 512_161_161_161_161_161-
                                 512_161_161_161_161_161-
                                 512_161_161_161_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C02-02-512_161_161_161_161_161-
                                 512_161_161_161_161_161-
                                 512_161_161_161_161_161.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C02-02-512_161_161_161_161_161-
                                 512_161_161_161_161_161.V50.sh.gz
```

7.7.2 Diagnostics patches for configurations with more than 5 units

For configurations with more than 5 units, a new file format has been introduced because of a length constraint in the file names.

See file naming formats for diagnostics patches in ***ZeBu-Server Installation Manual***.

```
zebu_V6_3_1.patch_0.diags_C02-10U_84481.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C02-6U_84481.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C02-7U_84481.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C02-8U_84481.V50.sh.gz
zebu_V6_3_1.patch_0.diags_C02-9U_84481.V50.sh.gz
```

7.8 EVE Vertical Solutions compatibility

The following sections list which versions of ZeBu transactors and ZeBu memory IPs are applicable with the present software version.

7.8.1 ZeBu transactors

The V6_3_1 software is compliant only with 64-bit EVE transactors and utilities.

These transactors/utilities can be requested from your usual EVE representative. The following table shows the latest releases of packages/utilities in a 64-bit version.

Doc Id	Transactor/Utility	Version (64-bit)
VSXTOR002	UART	2.9
VSXTOR003-2	JTAG XTENSA	1.7
VSXTOR003-3	JTAG + TAP controller API	1.3
VSXTOR004	SRAM SW	2.0
VSXTOR005	PCI Express	5.0
	PCIe Viewer	3.0
VSXTOR006	Ethernet (GMII)	2.1
	`etherPlug` utility	1.4
VSXTOR007	I2S	1.4
VSXTOR008	Digital Video	3.2
VSXTOR009	IEEE 1394 FireWire	1.2
VSXTOR010	MMC Device	1.2
VSXTOR011	AHB Master	1.6
VSXTOR012	Streaming Toolkit	1.7
VSXTOR013	USB Express	2.1
	USB Viewer	1.0
VSXTOR014	AXI Master + Slave	2.3
VSXTOR015	I2C	2.1
VSXTOR016	MMC Host	1.1
VSXTOR017	HDMI Sink	1.2
VSXTOR019	Video In	1.6
VSXTOR020	Ethernet Controller	1.1
VSXTOR021	ZDDRx (DDR2, DDR3)	1.0
VSXTOR022	TLM2 Adapter	1.1
VSXTOR023	KMI	2.0
VSXTOR024	SDIO Device	1.0
VSXTOR025	GMAC Ethernet	1.0

7.8.2 ZeBu memory IPs

ZeBu Memory IPs are all supported by V6_3_1 software (with specific licenses).

Doc Id	Memory IP	Version
VSIP001	ZDDR	3.3
VSIP002	ZDDR2	3.2
VSIP003	ZSDRAM	3.2
VSIP004	NAND Flash	2.0
VSIP005	ZMOBILEDDR	2.2
VSIP007	ZGDDR5	2.1
VSIP008	ZDDR_SP	1.2
VSIP009	ZDDR2_SP	1.4
VSIP010	ZDDR3_SP	1.2
VSIP011	ZMOBILEDDR_SP	1.4
VSIP012	ZLPDDR2_SP	1.4
VSIP013	ZLPDDR2_NVM	1.1

8 EVE Contacts

For product support, contact: support@eve-team.com.

For general information, visit our company web-site: http://www.eve-team.com

Europe Headquarters	EVE SA 2-bis, Voie La Cardon Parc Gutenberg, Bâtiment B 91120 Palaiseau FRANCE Tel: +33-1-64 53 27 30
US Headquarters	EVE USA, Inc. 2290 N. First Street, Suite 304 San Jose, CA 95054 USA Tel: 1-888-7EveUSA (+1-888-738-3872)
Japan Headquarters	Nihon EVE KK KAKiYA Building 4F 2-7-17, Shin-Yokohama Kohoku-ku, Yokohama-shi, Kanagawa 222-0033 JAPAN Tel: +81-45-470-7811
Korea Headquarters	EVE Korea, Inc. 804 Kofomo Tower, 16-3, Sunae-Dong, Bundang-Gu, Sungnam City, Kyunggi-Do, 463-825, KOREA Tel: +82-31-719-8115
India Headquarters	EVE Design Automation Pvt. Ltd. #143, First Floor, Raheja Arcade, 80 Ft. Road, 5th Block, Koramangala Bangalore - 560 095 Karnataka INDIA Tel: +91-80-41460680/30202343
Taiwan Headquarters	EVE-Taiwan Branch Room 806 8F, No. 20 GuanChian Road Taipei, Taiwan 100 Tel: +886 2 2375 9275